LEVEL II

UIREMENTS DEFINITION WITHIN ACQUISITION

AND ITS RELATIONSHIP TO

POST-DEPLOYMENT SOFTWARE SUPPORT (PDSS)

by

M. Hamilton & S. Zeldin

November 1979

HOS HIGHER ORDER SOFTWARE, INC.

NOTICES

DISCLAIMERS

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The citation of trade names and names of manufacturers in this report is not to be construed as official Government endorsement or approval of commercial products or services referenced herein.

DISPOSITION

Destroy this report when it is not longer needed. Do not return it to the originator.

(14) HOS-TR-22-VOL.-1

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: HOS TR-#22 Volume I
2. GOVT ACCESSION NO.: AD-A092967
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): (6) Requirements Definition Within Acquisition And Its Relationship To Post-Deployment Software Support (PDSS)

5. TYPE OF REPORT & PERIOD COVERED: (9) Final Report, Sep 1979 — Sep 1979

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): (10) M. Hamilton & S. Zeldin

8. CONTRACT OR GRANT NUMBER(s): (15) DAAB07-78-C-9026

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Higher Order Software, Inc.
806 Massachusetts Avenue
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: (12) 119

11. CONTROLLING OFFICE NAME AND ADDRESS:
Procurement Directorate USACERCOM
Fort Monmouth, New Jersey 07703

12. REPORT DATE: (11) November 79

13. NUMBER OF PAGES: 227

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report):

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for Public Release

18. SUPPLEMENTARY NOTES:

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

requirements definition, acquisition, PDSS, control, management, design, implementation, verification, documentation, integration

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The acquisition process is being viewed, here, as a system, in that an attempt is made to define its functions as well as the relationships between these functions. The resource allocation of the acquisition functions and the realization of these functions is viewed generically in much the same way as a software system is viewed. That is, whereas a software system is eventually resource allocated to a computer, the acquisition process could be resource allocated to a combination of people, organizations, hardware or software

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

393027

(the software itself resource allocated to hardware).

The long term goal of this project is to provide a formal definition of the acquisition phases and their relationships. We have now completed the first phase of this project where the short term goal was to understand more formally the relationships between the requirements definition process within the conceptual phase of the acquisition process and the post deployment software support (PDSS) system.

HIGHER ORDER SOFTWARE, INC.
806 Massachusetts Avenue
Cambridge, Massachusetts 02139

Technical Report #22

Volume I

REQUIREMENTS DEFINITION WITHIN ACQUISITION AND ITS RELATIONSHIP TO POST-DEPLOYMENT SOFTWARE SUPPORT (PDSS)

by

M. Hamilton & S. Zeldin

November 1979

Prepared for
United States Army Electronics Command
Fort Monmouth, New Jersey

PREFACE

It took years in the software engineering world, and there is still a long way to go, to realize that the major problems in developing software will not be resolved until the proper techniques are found to understand a problem before attempting to solve it. Certainly, technology has come a long way in that the concentration of efforts has moved away from better ways to talk to computers towards better ways to talk to each other before we talk to a computer. As a result, requirements and specification techniques have begun to capture the interest of the developers of large software systems, and even more recently the developers of systems in general. Such a system is the acquisition process of the Army.

Although the acquisition process is not currently envisioned as an eventual piece of "software," in that it must reside or be intended to reside within a computer, once developed, we can capitalize upon the lessons learned in developing large complex software systems since the definition of either type of system (i.e., software or others) is, after all, a definition of functions and the interfaces between them.

Following this reasoning, the acquisition process is being viewed, here, as a system, in that an attempt is made to define its functions as well as the relationships between these functions. To carry this software analogy further, the resource allocation of the acquisition functions and the realization of these functions is viewed generically in much the same way as a software system is viewed. That is, whereas a software system is eventually resource allocated to a computer, the acquisition process could be resource allocated to a combination of people, organizations, hardware or software (the software itself resource allocated to hardware).

The long term goal of this project is to provide a formal definition of the acquisition phases and their relationships. We have now completed the first phase of this project where the short term goal was to understand more formally the relationships between the requirements definition process within the conceptual phase of the acquisition process and the post deployment software support (PDSS) system.

ACKNOWLEDGEMENT

The authors would like to thank the following people for their invaluable contributions to this effort: A.R. Barnum, USAF, RADC/IS; D. Brock, USA, MIRADCOM; A. Campi, USA, PM, PLRS; A.B. Connelly, USA, USACC, Maj. W. Deutscher, USA, PM, PATRIOT; R.A. Fallon, USA, TRADOC; D, Gold, USN, Dir. Strategic Systems, PO; R. Kahane, USN, NAVALEX, BG; D. Lasher, USA, USACC; Maj. R.W. Mace, USA, USASC&FG; G.A. Mackay, USA, USACC; T. Madison, USA, FAS; Col. P. Mason, USA, USAADS, Col. A.B. Salisbury, USA, CORADCOM; Maj. J.H. Shroeder, USA, USAADS; T.A. Straeter, General Dynamics (formerly NASA LaRC); N. Taupeka, USA CORADCOM; N. Thompson, Systems Development Corporation, (formerly USA, CORADCOM); Ltc. W. Truehart, USA, CACDA; Maj. D. Whitfield, USA, CACDA.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

SECTION 1.0

INTRODUCTION

1.0 INTRODUCTION

Acquisition of systems can be defined as the obtaining of a capability, such as a technique for accomplishing a task; a device which performs work or duties and which serves as an extension of a person or persons; or an ensemble of equipment which includes both technique and the capability of performing tasks based on the technique. A system is an assemblage of objects, for example, techniques and devices, united by some form of regular interaction or interdependence. Systems, regardless of their goals, have certain fundamental properties but some systems end up performing their tasks and achieving the goals set for them better than others. By better, we mean some systems are efficient, cost effective, capable of responding to change over a long haul of service, and form the foundations for constructing subsequent generation of systems as improved techniques and machinery become available.

If the concept for a system includes what it is intended to do and the technology required for it to work is clearly described and defined, the procedure for acquiring the system is off to a good, but not necessarily adequate, start. This is because the procedure of acquisition of systems itself must be orderly, manageable, understood, defined and supported by techniques and devices. The procedure must be systematic.

The acquisition of a system is to some extent an attempt to combine the well known with what may not be known. Otherwise, the Army could simply purchase the system finished and combat ready from a suitable vendor. Thus, there is a definite risk. To control the risks, to identify them as early as possible before costs rise, the Army proceeds through a definite set of sequences of steps, pausing briefly at the end of certain sequences to allow review on progress, feasibility, and cost. As is evident from the public record, it is not always obvious when one step, such as advanced development, is really finished and a second, such as engineering development, is ready to begin with an assurance that engineering development is going to succeed.

This is because the acquisition of systems is very much like basic research in that it is hard, if not impossible, to determine ahead of time what is going to work. Unlike basic research, the acquisition of systems ends up or is intended to end up with a controlled, understood product which must work in the field. For this reason, and because large sums of public money and the nation's security are deeply involved, what is going to be difficult and what is going to be easy to do must be identified clearly and early in the acquisition.

To the extent that the systematics of acquiring systems is rigorous, controlled, defined and capable of allocating available resources and when and how they are needed, so will the success of the system being acquired be assured. A successful acquisition of a system is clearly distinguished from a worthy system of acquisition. In the case of a system of acquisition, "worthy" is defined as the capability to decide whether or not development should proceed. Success, then, is the same for the decision not to proceed and for a decision to proceed, because both are based on the same rigorously defined, consistent, and reliable criteria. Both types of decisions in the system of acquisition are then positive ones. They are positive because they breed excellence and build confidence for citizens, soldiers, officers, managers, administrators, and Congress.

A system has several parts called components; the system of acquisition contains several parts, such as the definition of requirements for software, the design and construction of the software, the definition of what the software supplies, the hardware (machine), the design and construction of the hardware, and the functions of the personnel. The parts of the software impinge and affect each other; the software links with machines and affects each other's performance. People both defining and using systems impinge on each other and parts of the system. These meetings and influences of one part of a system on another are called interactions. The process of defining how and when certain parts of the system should be brought to such meetings is a component of the system of acquisition; this component in turn can

influence and impact on the meeting of other components and these too are interactions.

The key, then, is to define a step in such a way that its definition is consistent with the definitions of other steps so that when any number of steps are required to join, they will be designed to join. Thus, the system of acquisition is modular, consistent, controllable in the same way the device being acquired is supposed to be.

SECTION 2.0

STATEMENT OF THE PROBLEM

2.0 STATEMENT OF THE PROBLEM

The process of systems acquisition to meet the tactical and strategic goals of the Army is intended to be timely and yield combat suitable products. There is, however, a rather clear record of failure for many system acquisitions. Time slippage, cost overruns, failure of the Army to fully specify what is required, contractor(s) failure to meet specifications of systems required for combat, and failure of the Service and contractor to deal realistically with new technologies are but a few episodes in many attemoted acquisitions [1,2]. Congress feels frustrated and leans harder on the services to justify budget commitments [3]. The services compete for the limited funds, at the same time trying to meet their obligations to the American people and to sustain a war-ready capability. As combat-related requirements become increasingly demanding, the time required for an acquisition may reach an average of 9-12 years, if current practices continue [4].

Appendix I summarizes the findings of a questionnaire developed for this particular project. Various combinations of the problems indicated in the responses to the questionnaire are plaguing the DoD in current systems-design and development activities.

2.1 Environmental Features of Systems Acquisition -- Changing Technology

One of the important features about the environment of systems acquisition is that the technology, available both for possible utilization in a system of the Army and for possible utilization by a potential enemy to counteract our systems, is in constant change. A system started five years ago thus can be combat-obsolete by the time it is ready to be deployed [5,6]. A major issue then is what, when, and how to insert new technology and yet have the system available for use to assure tactical and strategic advantage. It would be helpful, indeed, if the evaluation of a new technology for utilization in a system in development could be effectively made at the right time with the appropriate pay-off.

2.2 Inadequate Specifications and Consequences

A second important feature about the environment of systems acquisition is the disturbing fact that user requirements for a system are not always appropriately specified. By appropriately specified, we mean that what the Army wants a system to do is not described in a way that constrains and directs the contractor and assures the Army that its tactical and strategic goals can be realized. It is not surprising that testing and evaluation falters if the original specifications being tested were more detailed than required or too vague where detail and clarity are required for maximum performance [7,8]. It is evident that a tool which allows for the formulation of full and complete specification for a system is required. As specifications become clear, complete, and appropriate and not merely adequate, so will the Army's ability to acquire systems by fixed price rather than by cost-plus incentives. A contractor cannot justify time slippage, cost overruns, or failure of a system if the performance specifications are crystal clear at the very beginning of systems acquisition. The Army and contractor can then assume productive supportive roles with the specific interests of each having a good probability of being realized. But as noted earlier, in the real world, requirements often change. A new technology may appear, a recently recognized multi-service need becomes apparent or a new threat can occur at any time. Political pressures from Congress often ensue. The Army must be in a firm position to lay out the consequences of changing requirements. By fully specifying requirements in a modular way, the effects of changing them at particular phases of acquisition can be assessed before making alterations which may prove wasteful.

A closely related problem persistently haunts acquisition efforts. That problem is that the implementation or actual construction of a system does not flow or evolve directly from requirements and specifications. Indeed, specifications are often too vague to allow clear visualization of the system so that the engineer begins by filling in the holes and proceeds with a redesign which may or may not lead to a useful product [9]. Indeed, engineers often begin with

what they consider a significant subsystem, entranced by the apparent technical efficiency and performance characteristics of their device. The subsystem becomes the system for them and the rest of the system is built around their device. This is in effect a redesign process and may yield an elaborate device which is inappropriate for the tough demands of combat [2,5,10].

The Army acquisition process involves several contractors to take full advantage of the technical capabilities available in American industry. The problem is determining how and when each contractor assignment should begin and end. A process of complete and satisfactory specification is required so that each step in the development process is revealed and its relationship to other related steps made visible. With such relationships clearly established, the fitting of contractor expertise can be directly made with the system development.

2.3 The Quality of Management

A third important feature of the environment of systems acquisition is the character, indeed, the quality of management. This can be considered the crux of the matter of acquisition because people make decisions about technology, people administrate the process to a completion, and people make the qualitative and quantitative evaluations about performance of both devices and people who use and make devices. The public record is replete with instances of management failure to detect a serious flaw in a system meant to be used in demanding terrain and weather [2]. The public record is replete with instances of management failure to recognize at the right time just how untested a technology was for use in a communications system [1,5]. The public record is replete with instances of management changing requirements of the original or altered goals [11]. The public record is replete with instances of such rapid management turnover rate that it results in loss of management control [5]. The public record is replete with Congressional committee and DoD Directives describing in paragraph after paragraph procedures for

system acquisitions without recognizing the real-time adaptations required of a management system for managing system acquisitions [7,11]. Unfortunately, there is no substitute for real talent and expertise in management. Army systems are technically demanding, but only rarely are managers with either technical and management skills or managers with an acute appreciation of the science and skills of engineering assigned positions [11]. The situation continues to improve, but the pace of improvement is exceeded by the technical demands of the systems required by the Army. The requirement for adequate managers is thus important not only for the significant person to person leadership, but also for the technical leadership to monitor the relationship between phases of the acquisition process.

There are some 116 steps in the army acquisition process [12]. There is indeed a persistent large system, always operating and yet to whatever extent possible interacts with the elements of each of the sub-systems involved in systems development and acquisition. Are there the same rewards and incentives to top, middle and lower management to call a halt to any step of the process when it should be called as there are to presumed successful completions of so called milestones? How can a start or indeed a halt be called if no one really knows how to, when to, or what effects of a go or halt decision really are? Without this capacity to assess the what, when, why, and how, there can be few incentives for talented, aggressive management.

How can responsibility for error and success be assumed by anyone when there are no meaningful specific early warning signals for error detection available as management tools?

2.4 The Acquisition of the Army System

Surprise and concern have been expressed that so-called large systems such as Polaris, Apollo, and the Manhattan Project are successful in

terms of time of delivery and cost and performance, while so-called small systems, less complex, such as communication and vehicle development falter [13]. This criticism, however, is not justified. The record indicates, and as mentioned above, that there is, in fact, a large system superimposed over and interacting with the development of specific systems, i.e. the Army system [14,15,16,17]. The Services are engaged constantly in the development and acquisition of new systems in their efforts to meet their responsibilities. Indeed, as many as several hundreds of systems are in progress, each at various stages of the acquisition cycle at the same time. These and the field-ready products must be knit together. The coordination required to meet changing strategic and tactical demands must be sustained over a very long time period. The acquisitions of systems by the Army thus are considered as a "large complex" process rather than small and simple as is often claimed [13,18]. It is very likely that perception about system development for the Services arises from the difficulties in understanding the nature of the acquisition system.

2.5 Systems Acquisition -- Need for Understanding the Basics

It is worth emphasizing here that while it is possible to identify major environmental features of systems acquisition and certainly worthwhile, indeed necessary, to attack the deficiencies, there is a more fundamental issue which must be addressed. That fundamental issue is that the process of systems acquisition is itself a systematic process and yet it is not known what criteria or what qualities that system should possess. That fundamental issue is that the acquisitions system itself has never been described as a formal system. It has been <u>assumed</u> that acquisition is well enough defined that it can be described. But it has been only since the Second World War that the process became institutionalized. The precepts were defined on the basis of past experience and making not unreasonable guesses about the near future demands. A vocabulary has evolved, but judging from what Congress, managers, and contractors act upon, there is no evidence of a stable

dictionary [see Appendix I, Part 1, Section 4]. Because their has been permissiveness in allowing jargon to be substituted for language, important concepts are not perceived. Acquisition of systems is a quite practical process which has become impractical because no one really knows what is really meant in a shared fashion. There is, in fact, no culture and without culture, chaos results [19, example].

For example, managers, commanders in the field, and members of Congress often identify "successful" and "flawed" management and systems intended for the Service. Occasionally, but not routinely, the lessons learned from a previous success or failure are used in the acquisition of new systems [see Appendix I, Part 1, Section 1]. In no case, however, has there been a systematic method for applying the lessons learned, and, as pointed out in the paragraphs above, a successful acquisition process still appears to be a chance event. One argument which can be advanced to explain this circumstance is that it is difficult to project in a rigorous way, and in detail, the future course of a new application of a well-established or innovative technology. On the surface, this argument seems similiar to that applied to predicting the successful outcome of so-called "basic" or fundamental research. The major point, however, is that management and contractors have a need to clearly distinguish between what is really new and innovative and the process of adapting their procedures of management with utilization of resources in real time.

Analysis of the acquisition of several systems [1,2,5,20] indicates that the management, both on the contractor and on the service side, probably knew in advance what areas were going to prove difficult to complete due to technological problems or lack of previous application to the particular system under development. Yet they were unable to use this information in an effective way because it was unclear where in the system difficulty would first impact. As a result, the "wheel" is reinvented several times yearly in several different acquisitions. Along the same lines, there are many instances in which both the Services and contractor were unaware of the availability of appropriate

technology either because they were unable to search the scientific and technical literature in an informative fashion or because it simply appeared cheaper to redevelop a system component which had previously been developed for another application. Both of these ad hoc approaches lead to the familiar "we'll fix it up later when the time comes" approach, with the latter turning out to be too late, too expensive, and rarely a fix at all. Put simply, these events are due to the lack of a rigorously defined system of management for the acquisition of systems [1-5,7-11,21-23].

2.6 Current DoD Efforts

DoD is cognizant of many of the problems encountered in developing weapon systems. For example, in the software area the DoD Management steering Committee in July 1975 issued a Statement of Principles known as the Capstone Directive (which later evolved into DoD Directive 5000.29). The document seeks to make the role of software explicit, visible, and controlled in the weapon-system acquisition process.

The Committee also provided for studies [24,25] to be conducted on the state of software acquisition, development, and management for DoD weapon systems. These studies provide a strong statement of technical and management problem areas and supply recommendations on directions along which solutions may be found. The following quotation is from the MITRE study report:

> The major contributing factor to weapon system software problems is the lack of discipline and engineering rigor applied consistently to the software acquisition activities.

Studies [24,25,26] have shown that early and adequate planning is essential to the proper cost-effectiveness of weapon-system software. Planning is an intangible area which requires a large amount of

of money for "concept" rather than product. There is always pressure, therefore, to compromise this aspect of weapon-system development.

DoD is aware of the problems in cost, management, and technical development of software in general, and software for weapon systems in particular, as reflected in DoD Directive 5000.29. The net result of the policy stated in this directive is:

- computer resources must be managed in terms of elements which are integrated in an organized manner
- validation of requirements must begin at the front end (i.e., during concept formulation)
- specific milestones must be established to manage the life-cycle development
- software languages must be standardized
- software must be placed on a par with hardware throughout the acquisition process
- guidelines must be provided for DoD management and technical staff in developing weapons-system acquisition programs conducive to management and technical review
- qualified DoD staff must be provided for management and technical review
- software tools (HOLs and support software) must be provided for weapons systems development, thus reducing training, development, and review staff requirements
- a coordinated software research and development program must be established addressing critical software issues.

The environment which will be provided by DoD Directive 5000.29 will greatly facilitate the weapons-system acquisition process. MITRE, APL, CENTACS [27,28], and others [29,30,31], however, recognized that technical information, guidelines, management concepts, and directives alone will not solve the overall software problem. Flexible and effective software-development tools, within the context of a coherent, thorough and rigorous development methodology, have been perceived as the technology base necessary to surmount the problem.

Computer-based systems found in the military have counterparts in the commercial marketplace, including:

- Business application such as accounting and inventory control
- Scientific computation for research and development
- Large data-base applications with the requirement for data-base management and management information-system software
- Real-time, stimulus-response applications for command and control, communications, signal processing, and process control.

It is important to realize that commercial organizations, including computer manufacturers with their associated software development divisions and software vendors, are able to approach software and systems development with the advantages of continuity of personnel over time and identifiable common requirements for a particular set of users. For these reasons, it is possible for commercial organizations to develop a certain amount of applications-software support tools to aid their users in software and systems development. Examples of such software tools are higher-order languages/compilers, assemblers and macro-assemblers, linkage editors and relocatable loaders, libraries of common system-utility routines, libraries of common applications routines, data-base management and management information systems, and interactive man-machine interface support. Due to the availability of these software-development support tools, many users of commercial computer facilities and organizations have been able to realize cost benefits in their applications and systems-development efforts. These users share only marginally in the costs of tool development, but obtain the total benefit of each tool. It is clear that each of the tools (especially each common routine) used in a development project is simply one less tool to be designed, coded, and verified.

The DoD management policy and the collection of available and relevant software-development support tools can provide visibility and cost reduction in the area of systems development. The greatest savings in cost, with an attendant increase in reliability can be realized by incorporating the management policy and support tools within a framework of a development methodology which encompasses the total weapon-system acquisition process -- from requirements definition to deployment and maintenance.

DoD also has been cognizant for some time of the necessity of providing for review of progress made and of attempting to divide the acquisition of systems into particular phases to time reviews to resolve problems in system development. These reviews, the DSARCs, are intended to evaluate the progress made at each phase in the development of each part (component) of the system to assure that entry into a next phase is appropriate. According to the public record [32], only one system of all those sent for review has failed to pass its DSARC review in the last 2-3 years. Yet, failure of the systems passed through DSARC persists. Again, according to the public record, no colonel has ever been promoted in recognition for his decision to terminate a program before DSARC. It is clear that as in any other kind of risk-taking enterprise, there must be signals, signs, and evidence that termination is warranted before anyone will risk a career. In contrast to the Army, private enterprises have devices and personnel rewards, which makes a terminate decision about a system as professional and as acceptable as a go-ahead. Private enterprise, such as automobile manufacturers and computer firms, have invested in elaborate testing procedures and methods. DoD has made similar investments, recognizing that testing and evaluations are important crucibles for development of systems. While automobiles and computers are not often large, complex, expensive, and combat-suited as the Army systems must be, most private enterprises include provisions for linkage between testing and decisions to proceed in development. The pressure on such private enterprises to affirm an "acquisition" is great because profits, promotions, and sales volumes depend on having a product out there. The Public record suggests that the Army has yet to fully

exploit the possibility in rewarding equally termination and go-ahead decisions by career officers, in spite of the potential "profits" and required "payoffs" and "product" in terms of delivery of weapons that soldiers can use, cost-control, and Congressional receptiveness to new projects.

Efforts have been made to strengthen testing and evaluation, and recommendations have been made to make sure that more testing is not confused with excellent and meaningful testing [33].

The important key here is to define the requirements of a system at each phase of acquisition in order to identify testing and evaluation criteria with which to define the end of a phase and the beginning of another. While testing and evaluation are not the only evaluation crucibles for an acquisition, they can provide insight into the location and character of problems unknown in a system. The completed definition of a system offers the opportunity to distinguish more clearly between those components which require development testing and evaluation of prototypes from those which require operational testing and evaluation.

Past experience has indicated that it is the bringing together of known techniques and devices (software with hardware, hardware with hardware, software with software) which presents the majority of problems in system development and requires the largest share of time and money. In contrast unknowns, for example, a method to convert energy at the ultraviolet end of the spectrum to an image visible to the human eye or designing a landing craft (software and hardware) to make a soft-landing on the moon, requires less resources, perhaps different creative talents. In the acquisition of systems, the process should be made systematic enough to allow the immediate do-able to be done; this requires consistency of definition. Creative planners and designers can focus their efforts on the more blatant unknowns.

Thus the linkage between testing and evaluation and the requirements for each phase can form the basis of a sound, more reliable basis for determination of end-points and starts in the acquisition process.

SECTION 3.0

METHOD OF APPROACH

3.0 METHOD OF APPROACH

This effort was begun by making certain assumptions about the type of recommendations which should be made to the Army. These assumptions were formulated as a result of lessons learned from our own experience with other large complex systems. From these experiences an evolving list of system recommendations had been created which appeared to fall in the category of generic recommendations. But before finalizing these recommendations, data was gathered about the acquisition process and PDSS, in particular, for several reasons: it was necessary to understand these systems in order to relate to them, both for ourselves and for the Army; we did not want to assume a so-called generic property to be generic, when, in fact, it may not have been the case in this environment; and we wanted to determine if there were other generic properties missed as a result of previous analysis, but which might become apparent on the analysis of an even more complex system, than we had previously analyzed, the Army system itself.

This analysis of the acquisition process, with emphasis on PDSS, consisted of several types. Several documents concerning acquisition and PDSS were read. These documents varied from Congressional hearings to specific recommendations formulated by the PDSS working group. A questionnaire was then formulated which was intended to gather several and diverse viewpoints within the PDSS environment, other Army systems, and also Navy, Air Force, and NASA environments. Responses to these questions ranged from short "yes-no" responses to quite detailed and opinionated responses. Some of the questions were answered with no assistance and others were answered with the aid of the authors. A summary of questionnaire responses can be found in Appendix I.

Also, several members of the PDSS working group were interviewed, both on an individual-by-individual basis and on a group-by-group basis. The type of questions we were looking for answers to were: "What is the role of the material developer?" "What is the role of the combat developer?" "What is the relationship between the material developer and the combat

developer?" "What should it be?" Questions posed to members of the PDSS working group can be found in Appendix II.

The PDSS working group came up with several recommendations, which in philosophy, with respect to system design, correspond very closely to our own. We noted, however, that both the recommendations of the working group and ourselves in this area would be recommendations that would not only help to solve the PDSS problem, but also recommendations that would help to solve the problems of developing a system throughout its entire acquisition life cycle. There are other types of recommendations which were made by the PDSS working group that dealt with issues such as who did what functions, how many performed these functions, and where those functions were to be performed. PDSS members oftentimes held diverse viewpoints on these issues since each participant was concerned that his particular area was covered as best as possible. And often since more emphasis on one area would mean less emphasis on another, such conflicts were inevitable since resources in the Army are limited. Other than acquiring more resources in total, the only other way out was to find techniques which would be more cost effective for the Army system in general.

We chose not to deal with these "resoure allocation" issues on an individual-by-individual basis. Rather, an attempt was made to concentrate on those recommendations, which would impact PDSS and the acquisition process with respect to cost savings either on a system-wide basis, or on a subsystem-by-subsystem basis, irregardless of who was performing a function or where that function was being performed. For example, there are system definition techniques, which if applied, would eliminate the need for those tools that were needed only because those techniques did not exist before. The elimination of such tools eliminates the need for their maintenance, training in their use, and the need for disagreements concerning their use. For, in such a case, if there had previously been differences of opinion over where such a tool resided or over how many engineers were needed to maintain it, such a difference would no longer need to exist.

(Using the appropriate system definition techniques, an analysis can be made of an already-existing system environment to see what tools or modules are no longer in use and can be eliminated. We did just this sort of effort on one of our own system developments. The simulation environment was so large that it was difficult to know what was there. We found, in our analysis, that several modules and features were left over in the simulation environment from "assembly language days" when we were no longer using an assembly language. After discovering this fact, these modules and features were removed from the system. The result was that simulation runs were shortened and maintenance was no longer needed on an individual basis for these obsolete modules as well as with respect to the relationships between these modules and others in the system. Proper system definition techniques can also be used to determine where manual processes can be automated. On the same project where obsolete modules resided, twenty engineers were required to document software on a continuing basis over several years. The development of an automated documentation tool eliminated the need for this function to be a manual one, thereby freeing the twenty engineers to devote their time to other tasks.)

After gathering these types of information a set of high level checklists to be used by developers during the Requirements Definition phase was formulated. These checklists are presented in Section 4. Section 5 deals with how to make use of these checklists. Rather than recommend a particular methodology or a set of methodologies for defining requirements, we chose to concentrate on the properties a system definition should have. Checklists of a more detailed level were intentionally left out since these would involve a description of how to define requirements with the use of a particular methodology. Included here, however, are some examples of the use (and the description of that use) of the HOS methodology which demonstrates the overall principles discussed in sections that follow.

An HOS control map (see Appendix III) of the life cycle model was formulated in order to have a better understanding of the life cycle phases and the relationships between them. We did not have the time or the resources to

verify the details of the life cycle model. We did not consider this a serious drawback for our short term effort, however, since our intent was to concentrate on an approach for this type of application rather than on the specifics of the application itself. This preliminary analysis of the Life Cycle system is presented in Section 6.

Once a better understanding was obtained from these analyses our recommendations were formulated. The recommendations, presented in Section 7, concentrate on changes in the requirements definition process that would impact PDSS.

The analysis of the responses to the questionnaires was very helpful in attempting to understand the viewpoints of the people interviewed. More often than not, people agreed with each other in their responses. There was definitely agreement among users and developers that some drastic changes would have to be made in the area of defining requirements if the problems, so prevalent in the area of developing systems will ever be solved. Ideas, considered not possible only a short time ago, say five years ago, were readily accepted as the way to go in the future. As a result, we believe that recommendations such as those made here will be more readily accepted into the Army community than they would have been in the recent past. And since both users and developers are receptive to major changes for very similar reasons (e.g., the development process is plagued with problems because "the problem" is not well understood by anyone), why not take this opportunity to make these changes in a systemized and standardized manner in order to prevent the type of proliferation which got us into trouble before.

SECTION 4.0

CHECKLISTS

4.0 CHECKLISTS

Every top-level system developer goes through certain checklists before he begins to define his problem. This process is performed either consciously or subconsciously. We have found that an approach which explicitly adheres to a checklist is much more effective than one which does not. This is especially true at the requirements definition phase of a system development, since this is the phase where ideas are most fuzzy and therefore are more in need of clarification aids than other phases.

There are several types of checklists that are helpful in the very early system definition stages. They essentially serve as thought provokers for the formulation of concepts. These checklists are divided into major categories: the checklist of objects which should be considered and the checklists which show the step-by-step process to use in organizing those objects. The type of organizational processes that are used to create these checklists and the type of organizational processes that can be applied in the use of these checklists as well as the types of possible organizations for each checklist each serve as candidates for still another checklist (i.e., the methodology (e.g.,see [34]) with which to define a system).

This section concentrates on the types of objects to be considered in a system definition process. The first checklist is intended to support a top-level designer in understanding the types of systems that reside within his own system environment. The second checklist is intended to support that same designer in understanding the type of relationships which exist between systems within his own system environment. The rationale is that if a designer knows what the systems are in his own environment and he knows the relationships between these systems, then he will have a strong basis from which to understand his own system and thus be able to define that system in terms of its real environment. It is also possible that in the process of understanding the environment with

which he potentially has to deal that he will find, much to his surprise, that there are various systems in his environment that are, among other things, adversely affecting his target system, varying from bothersome to total disaster; obsolete and should be replaced or discarded completely; unnecessary or redundant; or incompatible with his target system. Then, too, he might find that something that should be in his system environment is missing and that its absence could have adverse effect on either its development or deployment. A third checklist is intended to serve as a goal for the types of properties that system definitions should have, no matter what the particular application is, for systems which are both reliable and cost effective in their development and their deployemnt.

4.1 The Target System Environment Checklist

When a system is verified, an environment or representation of an environment is created to subject it to or "run it" in. This environment is too often created "after the fact." And it often is created on an ad hoc basis, as is the system that is being developed. It is true that sometimes parts of an environment are created ahead of time, but this is usually for the purpose of determining certain parameters or constraints to be placed upon the system. And, it is not uncommon for the environment which is created for the purpose of determining these parameters to be quite different from the environment which is created later for the purpose of testing the system. Very rarely is there an attempt to define a system within a definition of that system's environment. Not only would such an approach suffice to evolve a more accurate system definition, but it would serve the other dual purpose of providing parameters and constraints and of providing the eventual test bed environment. Without such an approach, these three (at least) efforts are redundant, not coordinated, and they run a far better risk of not being able to perform.

A system is an assemblage of objects united by some form of regular interaction or interdependence.

A target system is a system which is to be defined and then developed. The acquisition process, as a system, is the development process of a set of target systems. The acquisition process itself could be viewed, however, as a target system with respect to its own development process.

A target system environment is that set of systems which will reside in the environment of the system that is deployed as well as that set of systems that are within the environment of the development of that system.

Every target system is developed within the context of several other systems. And each of these systems resides within an environment of its own, including the system which develops the target system (Figure 4.1-1). Each of these systems can be hierarchically viewed level by level with respect to its meaning as well as hierarchially viewed layer by layer, with respect to the realization of its meaning. The type of systems in the acquisition process which could inherently influence the development process of a particular Army system, and therefore, that particular system, are shown in Table 4.1-1.

Many of the systems that influence the development of a target system will also reside within the immediate environment of the deployed target system. These include those systems which are interoperable with the target system, machines performing the target system, the actual physical environment within which the target system resides, the real users of thetarget system and those deployed support systems such as an interpreter or operating system, which operate dynamically with the target system.

Not only must all of these systems be considered as the environment of the target system but also the environment within which all of these systems reside must be considered. The real users interface with real

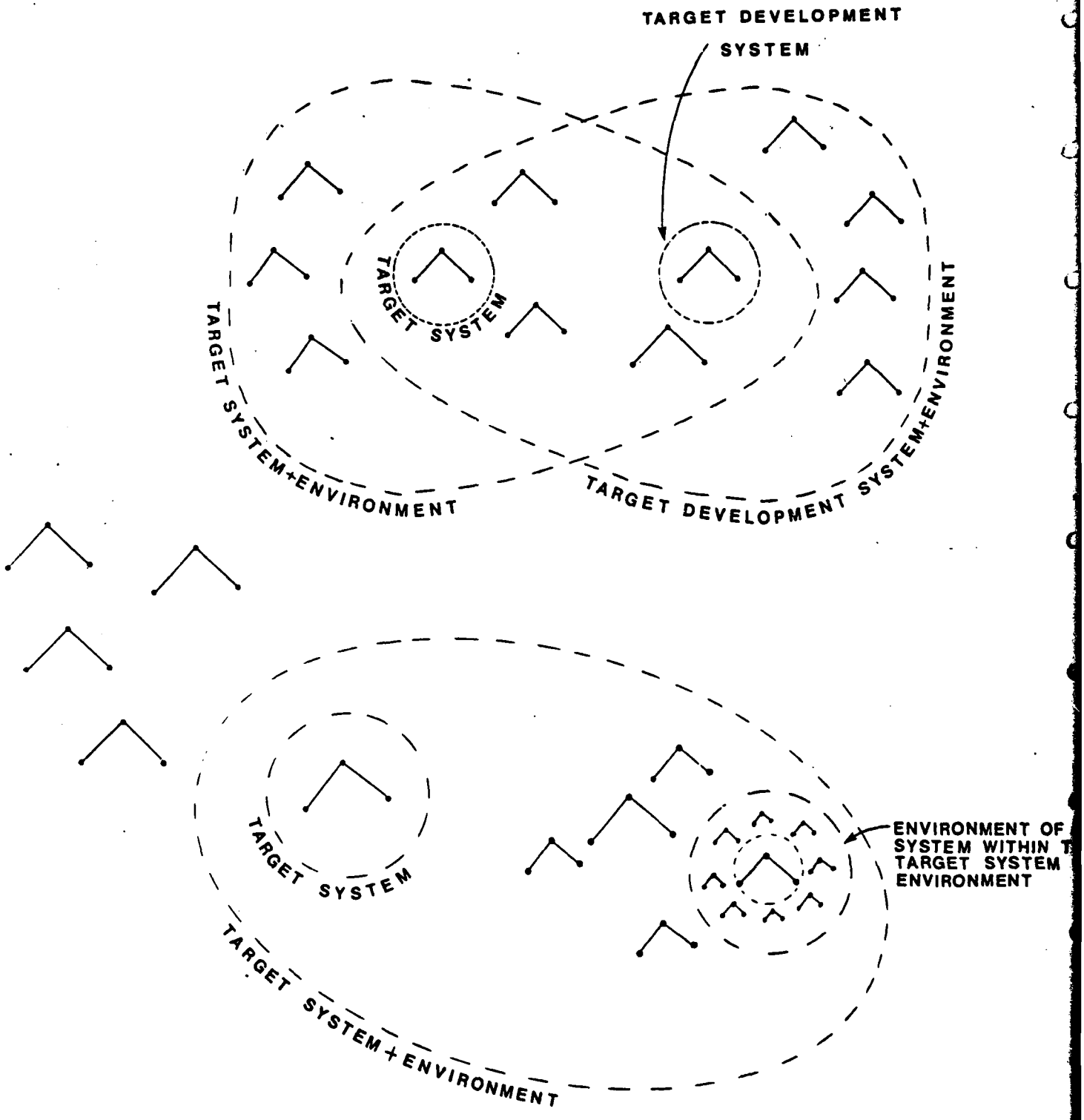


Fig. 4.1-1 Relationships of a Target System

Design - provides a definition of a target system.

Deployed Support Systems - the set of systems used during deployment to support a deployed target system (e.g., an operating system for a C^2 system).

Developers - the people or organizations who develop a target system, starting with the user model as requirements. These people include the material developers working together with the combat developers.

Development Goals - the set of overall objectives the development process of a target system should reach (e.g., accomplishing the development of the target system, efficiency with respect to development time, consistent interfaces among the developers and users.

Development Layers - the various forms of evolvement and viewpoints of those forms of evolvement of the target system. Development layers are created as a result of a discipline of development (i.e., management, design, implementation, documentation and verification).

Development Methodology - principles or procedures used to develop a target system (e.g., hierarchical definitions).

Development Process - the set of procedures, ordered with respect to time, used to put together a family of systems, within which the target system resides (e.g., communications systems).

Development Support Systems - the set of systems used within a development process to develop or help to make decisions about a particular target system development (e.g., a compiler for development or a graphics summary printout tool to aid in making a decision). These systems either support a development discipline or they replace it.

Development Training - those systems which are used to instruct developers as to how to develop a system.

Documentation - provides a description of a target system.

Host Machines - those machines which are used to represent the real machines which are used to execute the target system.

Library - a set of retreivable mechanisms used to define and develop a family of target systems.

Logistics - that system which concerns itself with resources, such as money and personnel, and the management of these resources of the target system.

Machines - those systems which will execute the target system. These could be radars, computers, people, or even an algorithm of which the target system could be an instance. The machine transforms the viewpoint of the target system from one of data to one of function.

Management - provides a chain of command, or control, for the development of a target system.

Physical Environment - the concrete surroundings within which a target system is deployed. For an avionics system, for example, this could be an aircraft, the universe, sensors, etc.

Real Users - the people or organizations who need a target system, and once acquired, will be operating the target system (e.g., the actual troops in various field site locations.

Real User Interface - that system which is developed to simplify the interface between the user and the target system.

Resource Allocation - provides an implementation of a target system.

Simulated or Emulated Target System - a representation of the target system which could vary in degree of accuracy with respect to the real target system. A computer as a target system might, for example, be simulated or emulated on another computer before it is developed. In this way, design errors that are found "after the fact" are cheaper to fix.

Table 4.1-1 Target System Environment Checklist

Simulated or Emulated Target System Environment - a representation of the target system environment which could vary in degree of accuracy with respect to the real target system environment. In some cases, the target system environment has to be simulated or emulated since the target system cannot be tested in its actual environment until its real mission (e.g., APOLLO or a nuclear war).

Target Development Interoperable Systems - systems which affect the target system development during the deployment of the development. These include systems which communicate with the target system developemnt.

Target Development Politics - the set of human conditions, such as agreements or conflicts, which inherently influence how a target system is developed.

Target Development Process - the particular set of procedures, ordered with respect to time, used to put together a particular target system. Whereas, for example, the development process may determine allowable relationships which are in common for a family of functions (e.g., an approval process), the target system development process may determine the specific functions to be performed within that approval process (e.g., approval process for a particular module of software on a particular computer).

Target Development Technology - the state of the art or known state of the art which helps to determine what is possible to accomplish in the development process of a particular mission (for example, the length of time that is needed to develop an effective fiber optics system).

Target Goals - the set of overall objectives a target system should reach once developed (e.g., accomplishing the mission, efficiency with respect to execution time, consistent interfaces among the subsystems).

Target Interoperable Systems - systems which affect the target system during its deployment. These include systems which communicate with the target system (i.e., systems which provide input to the target system and systems to which the target system provides output).

Target Politics - the set of human conditions, such as agreements or conflicts, which inherently influence that which is required for a target system and therefore what it becomes.

Target System Development Properties - a set of characteristics, which inherently exist in every module of the target system development process.

Target System Properties - a set of characteristics which inherently exists in every module of the target system (e.g., controlled hierachy of priorities, traceable data flow).

Target Technology - the state of the art or known state of the art which helps to determine what is possible to accomplish for a particular mission (e.g., fiber optics technology).

Training - those systems which are used to instruct the real users as to how to operate the target system.

Tests - a set of scenarios used to create or represent the interoperable environment (including the real user) of the target system in order to examine that target system.

User - the combat developer.

User Model - the target system as initially defined by the user, but later refined usually as a result of material developer and combat developer interactions in conjunction with other agents such as the logistician, trainer, etc.

Verification - provides an execution of a target system.

Table 4.1-1 Target System Environment Checklist (continued)

users of other target systems (e.g., users of ground systems with users of air systems). The user interfaces with other users (e.g., Army, Navy and Air Force). The development process could be used to develop systems other than the target system, or it could be interfacing with development processes of other systems. The disciplines could be used to develop systems other than the target system, or they could be interfacing with disciplines used to develop those other systems. The support systems and library system could be used by systems other than the target system. The goal system resides within a larger set of goals, and, of course, the same is true of the political system.

Before we discuss the various types of relationships between all of these systems, consider:

- All of these systems are vulnerable to change. New requirements can be added to these systems as a result of an error or because it may be desirable or necessary to add a new feature. That new feature could be a change in capability to a target system or a change in personnel within a system of disciplines.
- All of these systems are dynamic. Not only does it make a difference with respect to which instantiation of a system interfaces or influences another, but also the timing of such an instantiation, as an event, could make a difference as well.
- Many of these systems are related to each other in terms of feedback where the effect of one system on another system in turn results in each system eventually affecting itself.

Knowing the target system environment checklist is not only useful in determining what should go into a definition or who to talk to in determining a set of requirements, but it also can be used in determining the effects to the environment of a change to a target system or the effect to the target system of a change to the environment. For example, a change in a target system could obsolete a part of a simulator or certain procedures of the user. Conversely, a change to the physical environment could make obsolete a part of the target system. Thus, whenever, a change is contemplated, a very useful exercise is to walk through the environment checklist in order to help determine the impact of that change, and then the feasibility of making such a change.

Just having the ability to know all of the systems in the target environment is a powerful tool. But that tool can be even more powerful, if the relationship between these systems are better understood.

4.2 Checklist of Target System Relationships

There are many types of relationships among the systems (systems themselves) in a target environment, including those systems which exist in the development process of that target system. Knowing these relationships ahead of time can make a significant difference in the way a target system is defined and developed. In fact, there are times when such a difference could determine whether or not the target system will even work. It could also determine to a large extent both savings in time and resources, both in the development of a target system and its deployment. Knowing the relationships between the various systems in the target system's environment can also help to determine (more explicitly than just knowing the existence of these systems) when a change to one system should result in a change to another. Knowing these relationships can also help to determine which systems should be added or which systems should be deleted in order to make a more effective target system environment. Again, a life cycle savings in time and money. Table 4.2-1 lists some of these relationships.

A preliminary analysis of the systems within the target system environment and their relationships is shown in Table 4.2-2. This preliminary attempt to understand the types of relationships which exist between the systems in the target system environment and the target system itself has helped to identify why the development of a large complex system is neither easy nor well understood.

Table 4.2-2 can be used in several ways. For example, a combat developer may want to change the requirements in the target system. The checklist can be used to determine what systems in the target system

Analysis - When one system evaluates another system for whatever reason, whether it be verification, documentation, management, implementation, or design, that system performs an analysis on the other system. This could be performed statically or dynamically. Thus, each of the disciplines of the development process, for example, performs an analysis of one kind or another on the target system.

Behavior - When one system defines the properties of another system with respect to the relationships between that other system's objects, that system is a system of properties with respect to the other system.

Change - Sometimes one system could alter the use of objects, and their relationships, within another system. For example, within the disciplines of development, a designer could insert data, functions or structures within a target system. Or, he could delete or change these objects as well. A documenter, on the other hand, might wish to change the description of a system to make it easier to understand.

Communication - A system which communicates with another system either sends it output or receives its input, or both. In the latter case the relationship between the systems is one of feedback. Such a relationship could exist between two Army systems such as PLPS and TDS. Or, managers of each of these two systems would communicate with each other. And, it is conceivable that the development processes of these systems might communicate with each other, as well.

Constraints - When one system defines a set of criteria that must be followed or limits that must be adhered to by another system, that system is a system of constraints with respect to the other system.

Control - A system which is in control of another system determines which functions of that system are to be performed, input and output access rights for its data, the ordering of its functions, the relationship between its input and output, and what it means to have improper input. Ideally, this relationship of control would exist between functions of, for example, the real user system and the user system, the user system and the target system. This is an example of static control with respect to the target system (Figure 4.2-1). The target system itself would have the relationship of control inherent among its own modules. This is an example of dynamic control with respect to the target system (Figure 4.2-2). The development management functions would have control of the respective development phases and thus the development layers of the target system. This is an example of dynamic control with respect to the development process of the target system and static control with respect to the target system itself (Figures 4.2-3, 4.2-4). A given target system can be checked statically for correctness with respect to dynamic control, but that same system has to be checked dynamically with respect to static control.

Decision Support - When one system is used as information to aid in determining what should go into a target system, that system is is a decision support system to the target system. Such a system can be used to determine, for example, whether or not a target system is a go or no-go situation or if it should be changed.

Definition - A phenomenon whereby one system states the meaning of another system. A designer is an example of someone who could provide a system definition. An automatic tool could provide a projection of a formal definition to become a new definition.

Description - A phenomenon whereby one system expresses another system. This description could be in verbal form, pictures, graphics, or even body language.

Derived Systems - Those systems whose definitions can be traced to already defined systems.

Equivalent - Two systems are equivalent if given that one of the systems is a subset of a larger system, a replacement of that subset system by an equivalent system would be in effect, no effect. That is, their overall behavior is, in effect, identical. And, sometimes one system which is an improvement over another system for either, say, reliability or efficiency reasons, might replace that system or not depending on factors such as goals, politics, time, the development process, etc.

Execution (Instantiation) - A phenomenon whereby one system causes another system to perform. A verification of a system is a form of a system execution. A system which is an instantiation of another system is one which makes possible the completion of one performance pass of that system. That is all data, functions and structures (relationships between functions) of one system have successfully been realized at a given instant in time by one of a possible set of instances which is that other system. Thus, the development of the target system itself could be viewed as an instance with respect to the execution of the development of a set of user systems. An implementation of the specification of that same target system could be viewed as an instance of the development of the specification of the target system. In such a case, an implementation of the target system is actually an execution of that particular development step of the target system.

Existence - Sometimes one system affects another system by the mere fact that it exists. Thus, for example, if a particular computer or library system existed, one might use it. If it did not, one would not. And, if two software systems were being processed within a multiprogramming environment within the same computer, the existence of one could affect the timing of the other.

Table 4.2-1 Target System Relationships Checklist

Implementation - A phenomenon whereby one system makes it possible for another system to perform. An implementation process takes place when one system assigns a name to an object or an object to a name for another system's specification. That is, the implementation system prepares another system to run on a particular machine environment. When such a process has been completed for a given system specification, than that system specification has at least one system implementation. That system implementation represents a "how" with respect to its specification (i.e., its "what"). Whereas an instance of a system specification represents an execution of a system specification, an instance of the process of preparing that system to be executed represents an implementation of that same system but an execution of the development of that system. Similarly, an instance of the implementation represents an execution of that system's implementation with respect to the system itself. Therefore, the specification and implementation states, each of which represents all instances of the target system, are static; whereas the execution state, which represents only one instance, is dynamic. A manager implements when he resource allocates, for example, people to tasks; a compiler implements when it resource allocates, for example, storage to tasks.

Interoperability - In [16], interoperability is defined to be the capability to directly exchange data in a prescribed format and frequency with mutual non-interference and process the data exchanged. Two systems are interoperable with respect to each other if the existence of one can affect the behavior of the other. Thus, systems which communicate with each other are interoperable systems. But two independent systems which are designated to operate in parallel could also have an effect on one another. Say, for example, if there were limited machine resources and one system had to wait for the other to complete before performing. Or, one system's output might communicate with a system which communicated with the second system. Again, if there is a decision to deploy one system or another based on certain criteria, the decision to perform one over the other certainly affects the other in that although both may have been in a state of readiness, the activation of one may require either a de-readiness, or at least a longer wait before the other eventually is activated.

Intersection - A phenomenon whereby one system is combined with another system to form a third system which is that part that is common to both systems. Thus, that part in a library which is used to form a target system definition is the intersection between the target system and the library. When more than one system shares routines within the library, there is an intersection between at least three systems forming yet another system.

Machine - Sometimes one system is run by another system. That other system is a machine with respect to the first system, Thus, a computer system, which could be a support system to the target system would be a machine, with respect to running the target system, although a development process itself could be viewed as a machine with respect to the running of the development of the target system. The machine essentially takes in one system in its static or "being" state, i.e., as data, and transforms it to its dynamic or "doing" state, i.e., as a function.

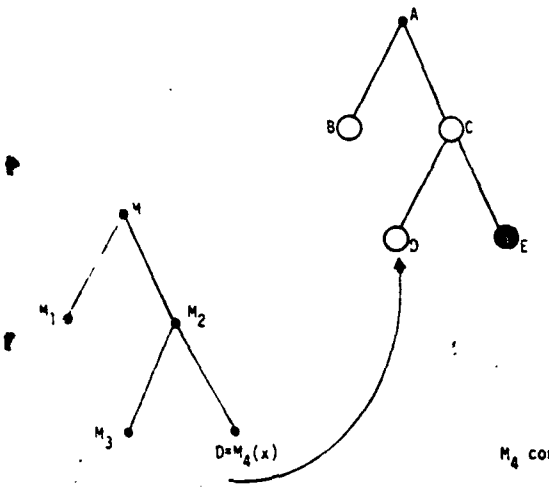
Projection - A phenomenon whereby one system definition corresponds, level by level and layer by layer, with a part of, or "slice" of, another system definition. For example, either a priority structure definition, only, or a data flow definition, only, could be a system projected from another system definition which integrated both the priority structure and the data flow.

Replacement - Sometimes one system could take the place of another system, either temporarily or permanently. A simulation of a system would represent a temporary replacement of a system. The firing of a manager or a designer could result in a permanent replacement. A back-up system in non-real time or real time could temporarily or permanently replace a primary system depending on the seriousness of the situation at hand.

Representation - A phenomenon whereby one system is enough like a target system to "speak" for it. There could be several representations of a target system. They could all be deployed, for example, on different computer environments and thus behave slightly differently due to the effect of other and varying systems in the same environment, as different CPU power, accuracy, etc. Whereas an implementation would, in its pure form, maintain the exact behavior of a specification, a representation could vary somewhat (e.g., a simulation which compromised accuracy to check out certain interfaces could be a representation and not an implementation). If a representation is accurate enough, it could eventually replace an implementation, but then the specification in essence changes, thus making the representation an implementation.

Subset - A phenomenon whereby one system is a part of another system. This could be a subsystem with respect to that system or, a collection of members of a data type with respect to that data type.

Table 4.2-1 Target System Relationships Checklist (continued)



M_4 controls D as data

Fig. 4.2-1 Static Control

A controls B and C as functions

C controls D and E as functions

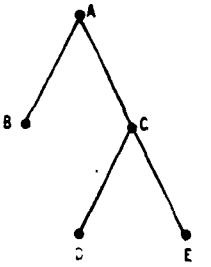
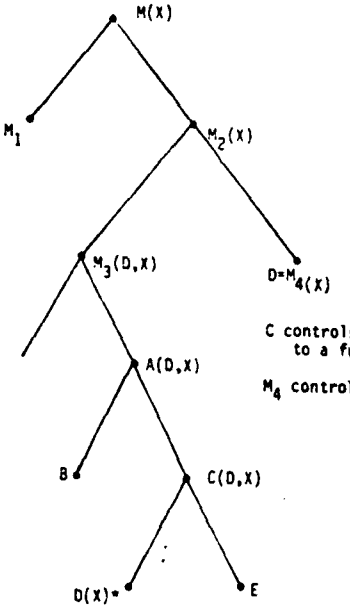


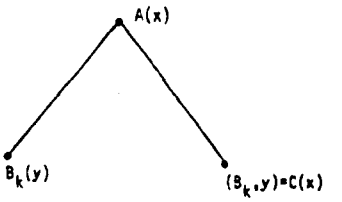
Fig. 4.2-2 Dynamic Control



C controls the transformation of D to a function

M_4 controls D as data

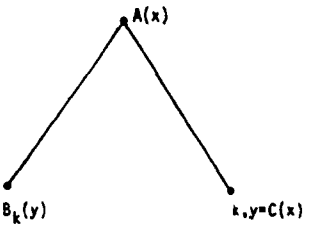
Fig. 4.2-3 Static and Dynamic Control



A controls C as a function

A controls the transformat of B_K to a function

C controls B_K as data



A controls C as a functio

A controls the transformat of K to a function

C controls K as data

Fig. 4.2-4 Reconfigurable Static and Dynamic Control

*Equivalent notation for a function, I, that applies D to X i.e., $I_D(x) = D(x)$

TARGET ENVIRONMENT SYSTEM	RELATIONSHIPS TO TARGET SYSTEM
design	definition, change, analysis
deployment support systems	decision support, interoperability
developer	control, definition, implement, execution, describe
development layer	subset, projection, constraints
development methodology	decision support (target system development), intersection (target system development)
development process	control, definition, implementation, execution, description, change, analysis
development support systems	decision support (target system development), interoperability (target system development)
development training	decision support (target system development)
documentation	description, analysis
host machines	machine, representation (target system machine)
library	intersection, derived, execution, machine, existence
logistics	decision support (target system development)
machines	implementation, execution
management	control, existence, analysis
physical environment	interoperability
real users	interoperability
resource allocation	implementation, analysis
simulated target system	representation, replacement, implementation, equivalent
simulated target system environment	interoperability
target development politics	decision support (target system development)
target development progress	control, definition, implementation, execution, description
target development technology	decision support (target system development), intersection target system development)
target goals	decision support
target interoperable system	interoperability
target politics	decision support
target system development properties	behavior (target system development)
target system properties	behavior
target technology	decision support, intersection
tests	representation (interoperability) simulation (interoperability), interoperability
training	decision support
user	decision support
user interface	decision support, projection, interoperability
user model	subset, projection
verification	execution analysis

Table 4.2-2 Possible Relationships of Target System Environment Systems to Target System

environment will change and how they will change, or if the change is even a feasible one, both from the standpoint of performance and of cost effectiveness. On the other hand, if one of the systems in the target system environment is being considered as a candidate for change, the effect to the target system itself can be analyzed in the same manner. Likewise, if it were considered desirable to understand more about a particular type of target system environment (e.g., decision support systems or interoperable systems), those sets of systems of a particular type could be singled out and analyzed to determine how effective or perhaps how compromising they really were to the state of the target system.

Major categories of relationships that influence the development of a system include the in-line target system subsystems (Figure 4.2-5) which are further explained in Figures 4.2-6 and 4.2-7; the target system viewpoints (Figure 4.2-8); the target system development process (Figure 4.2-10); the target system decision support process (Figure 4.2-11); and the target system development decision support process (Figure 4.2-12).

It became clear from this analysis that PDSS requirements, for example, should be considered almost immediately in the acquisition process, since the deployment support systems are interoperable with the target system, the real user interface is interoperable with the target system, and the real user is interoperable with the target system. And, the training of the users is highly dependent on the success of the real user interface. It is, therefore, not surprising to see that problems have resulted from the fact that PDSS is brought in too far down stream in the acquisition process. It is almost no different than bringing in the requirements definitions (now at conceptual phase) during deployment, since, in fact, the PDSS requirements are, in real life, a subset of these conceptual requirements.

It also became clear that there are, in actuality, two major developers, as their actual names would imply: the combat developer (user) and the material developer (developer). There is, therefore, a strong rationale for the user to include the developer in his original requirements definition phase, as well as the developer to include the user throughout the development phases of acquisition.

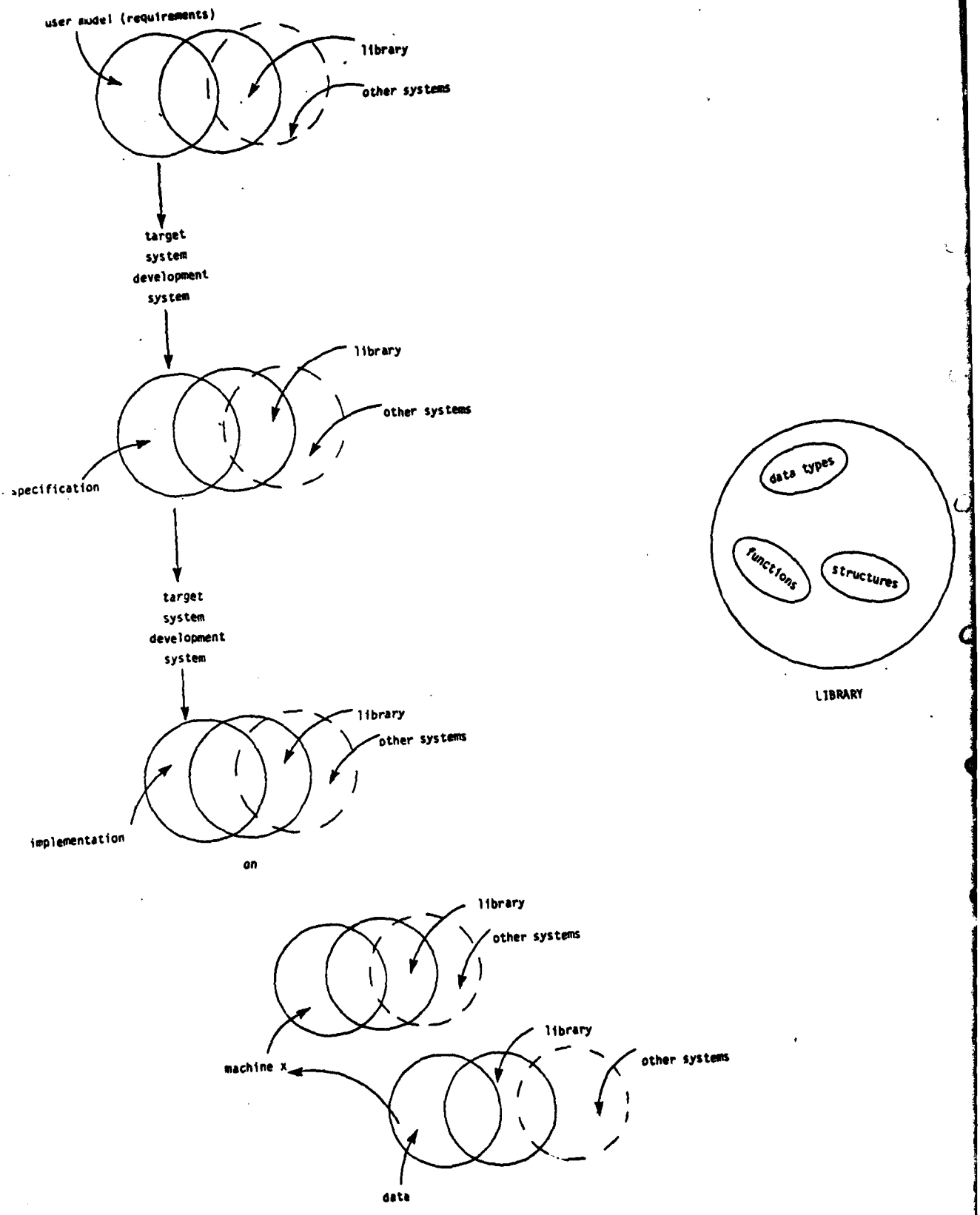


Fig. 4.2-5 In-Line Target System Subsystems

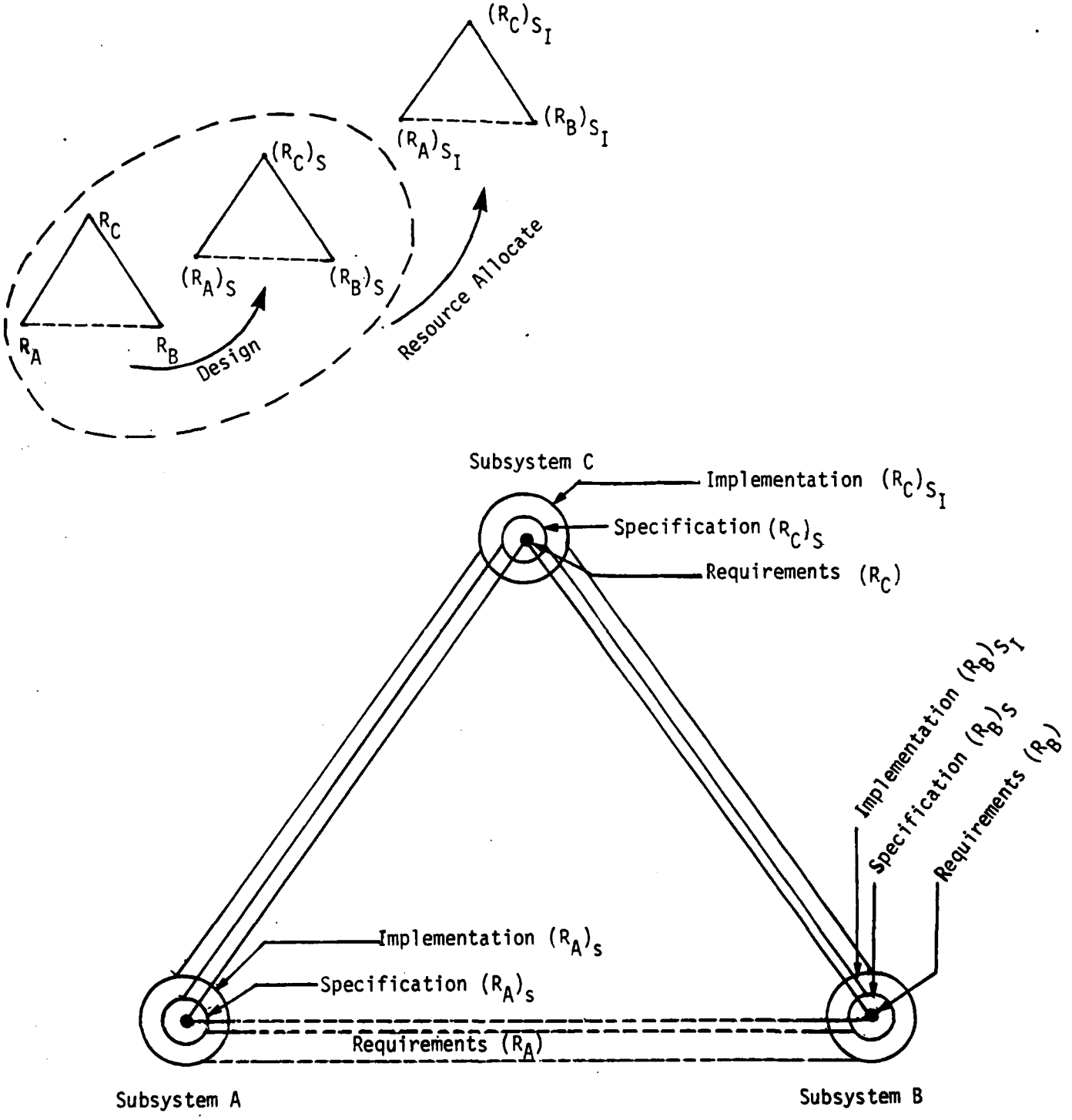


Fig. 4.2-6 Level-By-Level Integration of Subsystem A and Subsystem B Development Layers

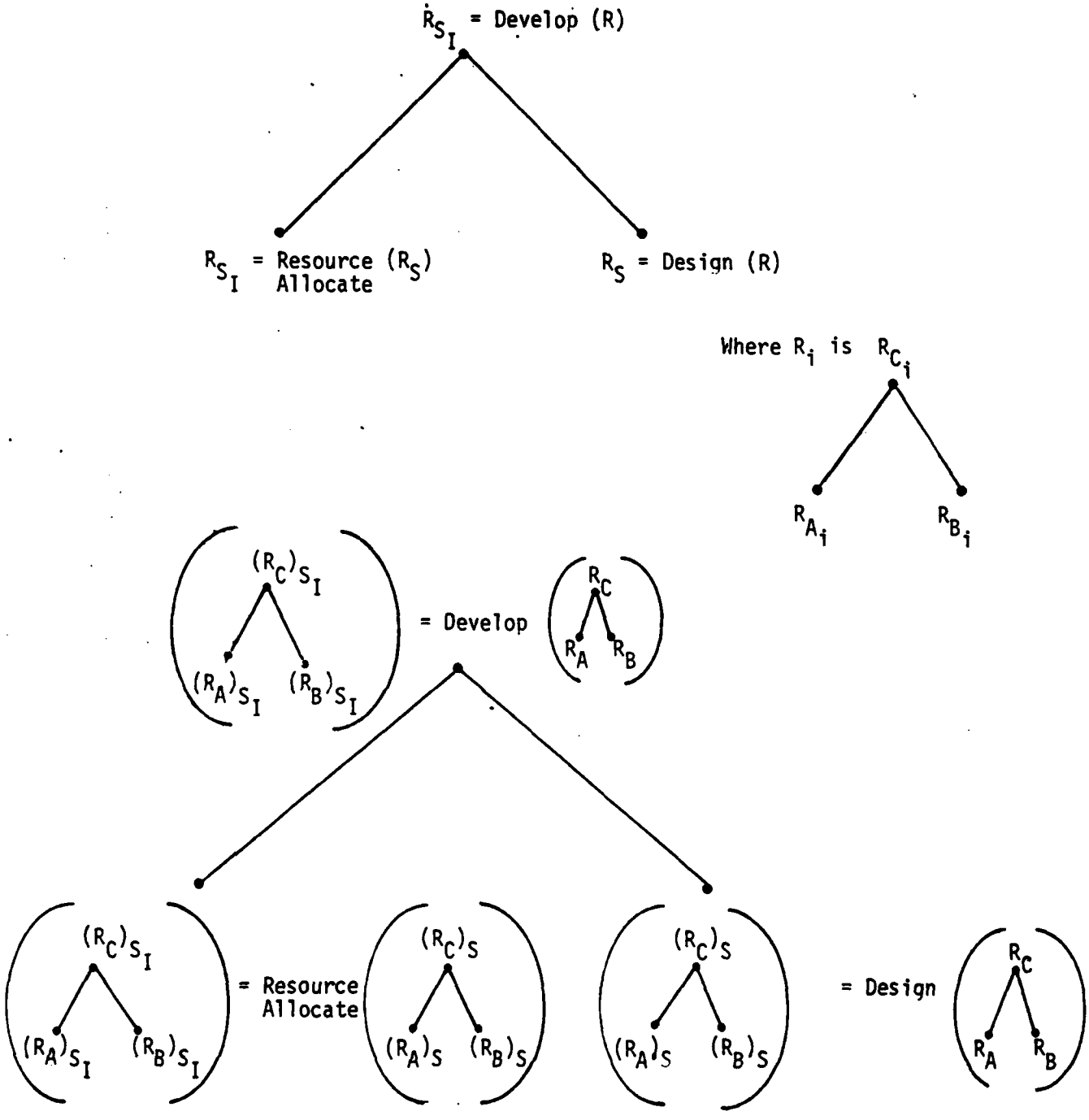


Fig. 4.2-7
Layer-By-Layer Integration of Subsystem A and Subsystem B
(Integrate from the front end)

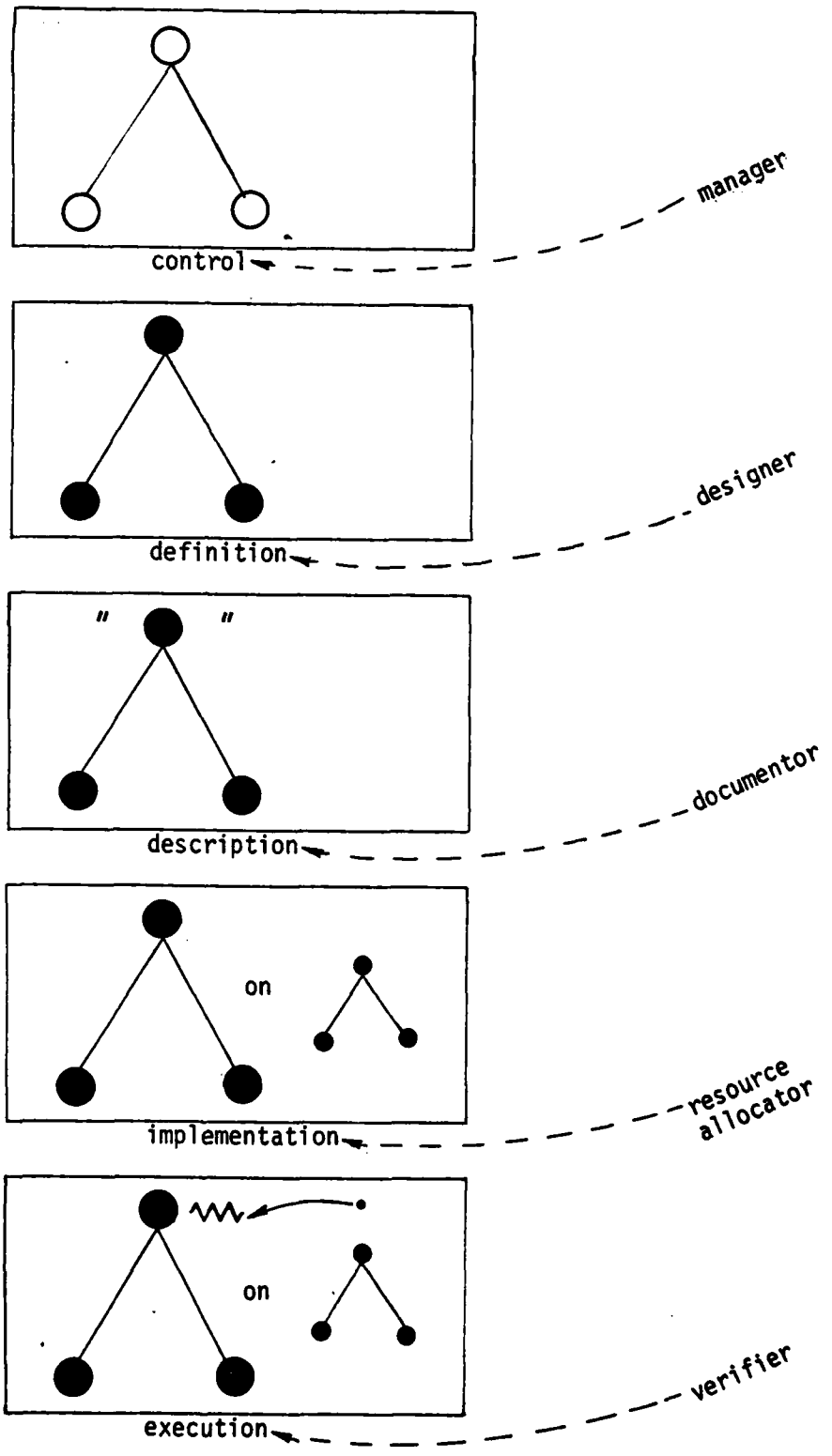


Fig. 4.2-8 Target System Viewpoints And Their Relationships To The Target System Developers

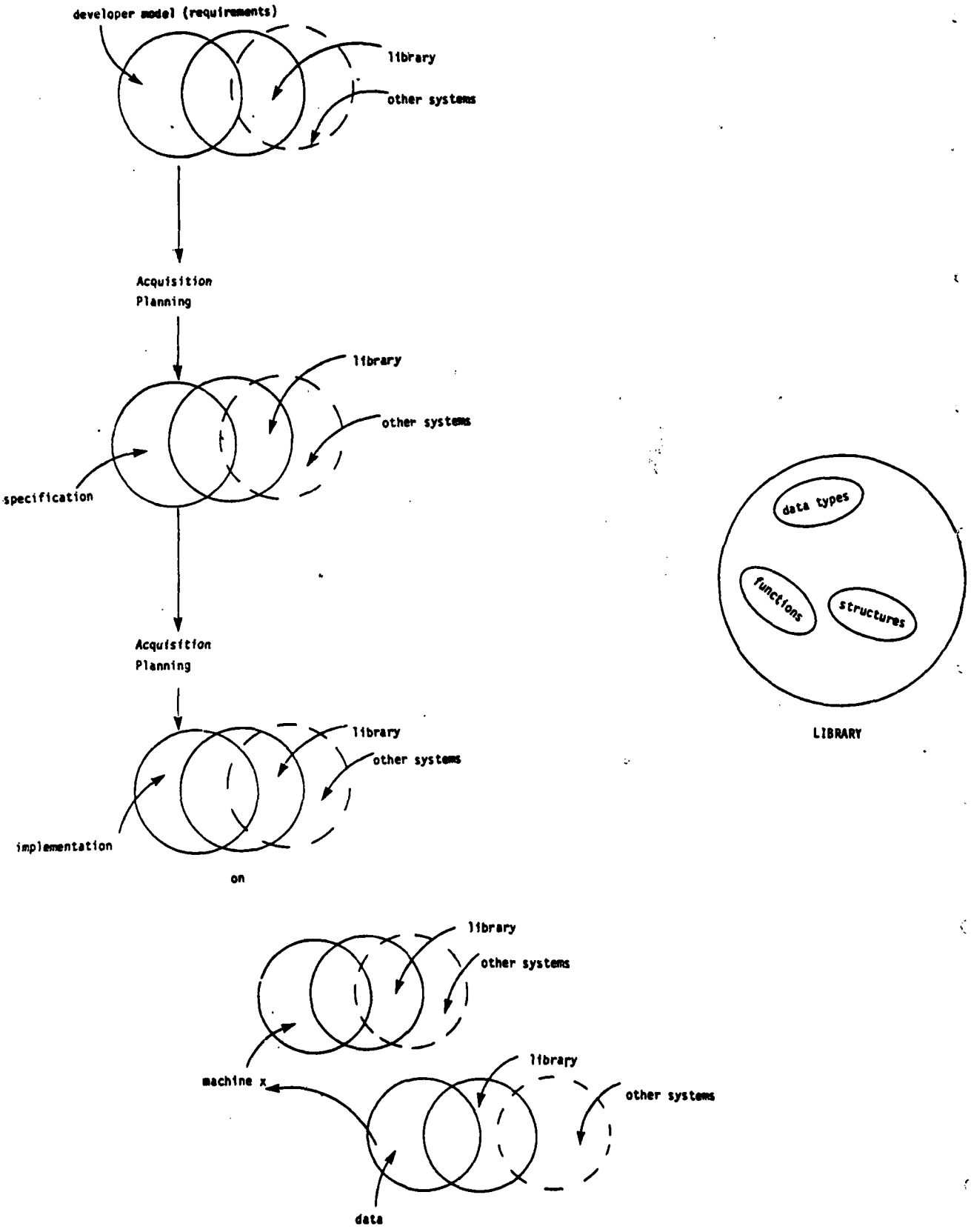


Fig. 4.2-9 Target System Development System

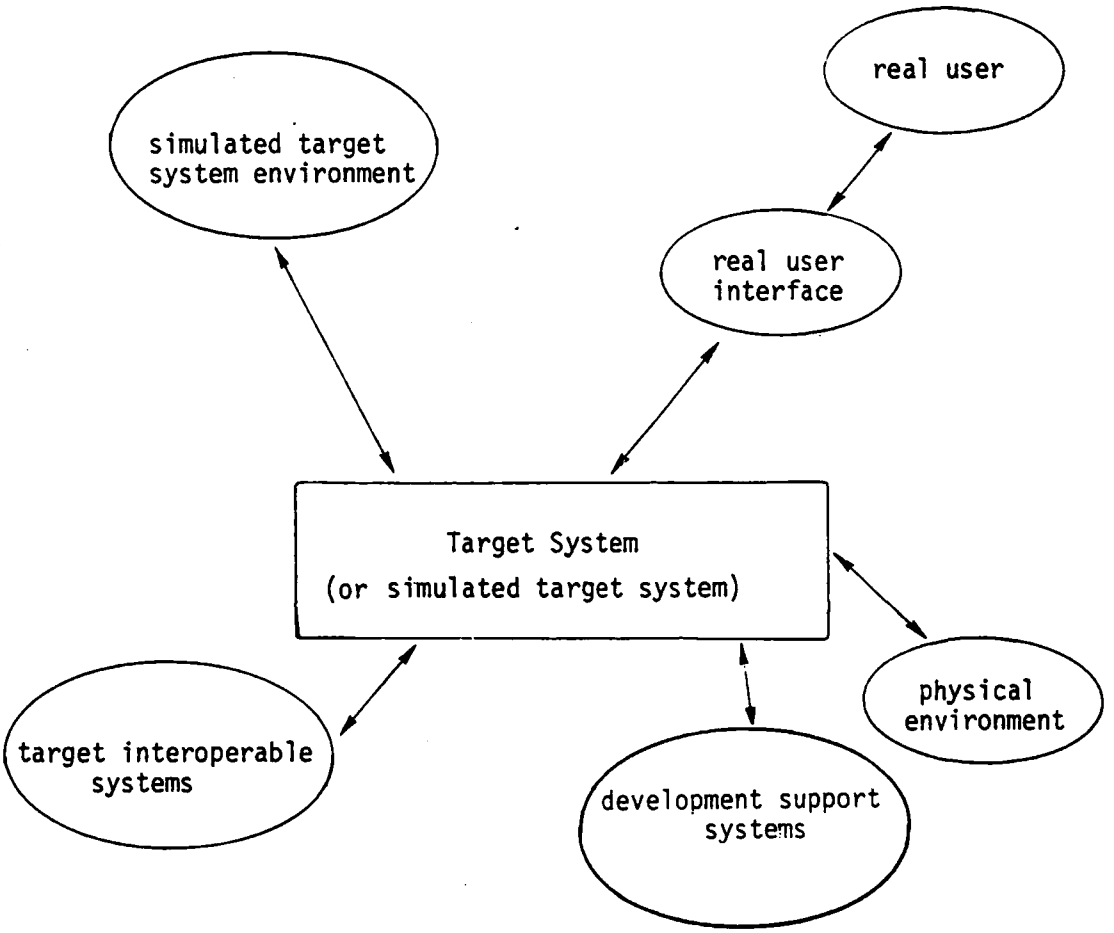


Fig. 4.2-10 Target System Interoperability (test environment)

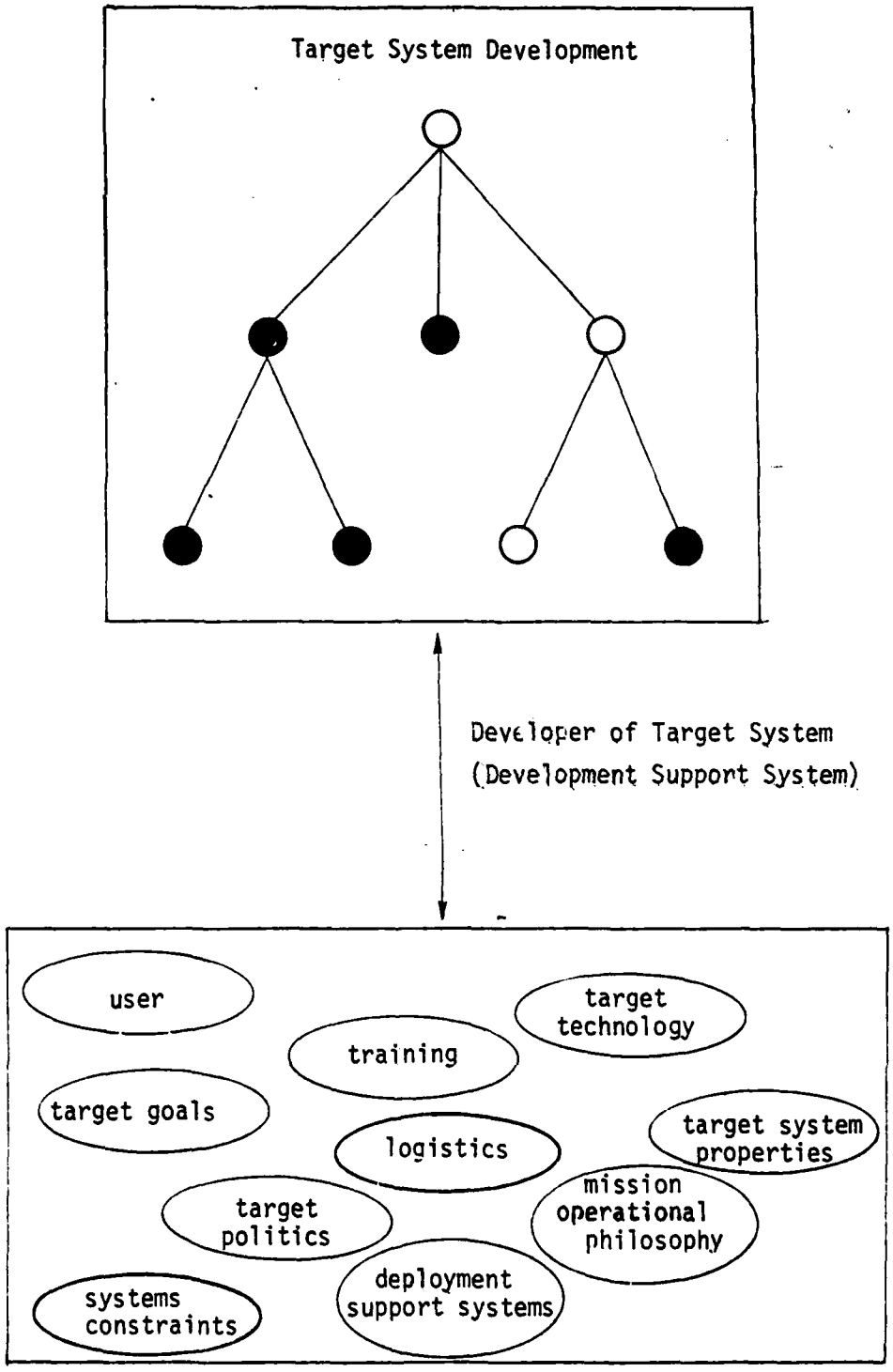


Fig. 4.2-11 Target System Decision Support System

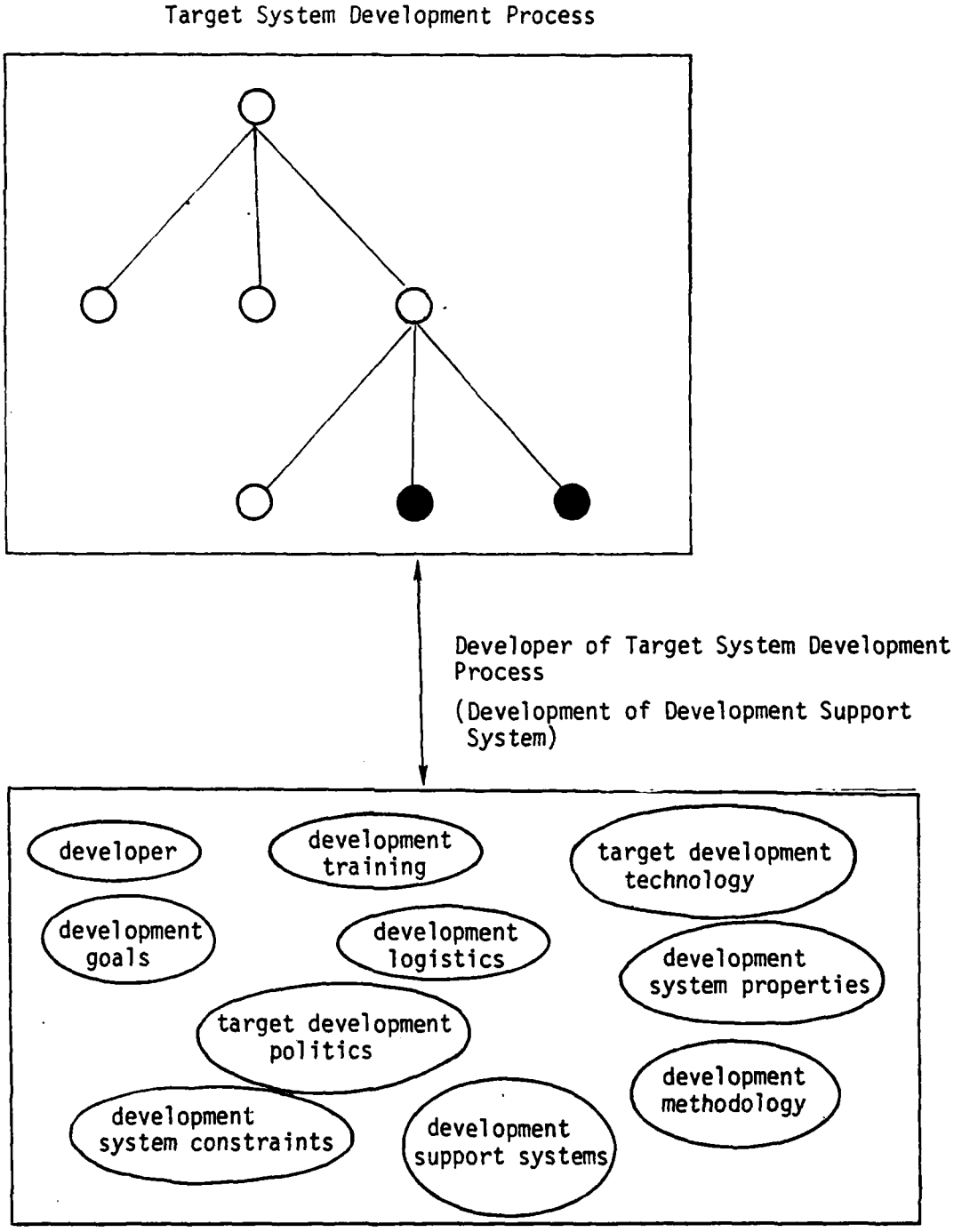


Fig. 4.2-12 Target System Development Decision Support System

4.3 Checklist of Desirable Properties for a Requirements Definition

The assumption is made, here, that in order to really understand a target system and its development, it is necessary to understand the target system environment and how its environment affects both the target system and the system of developing that system. The success of such a process (i.e., that of understanding) is dependent on properties of definition.

First the systems that exist must be identified. Then, the type of relationships that exist between one system and another system must be identified. Then, the target system would be defined within its environment as a system itself. Let such a system be called the target system and its relationships. Now if those relationships are under control, as is the ideal case, such a system will be called the target system and its relationships under control, TSRC.

In order to define TSRC, principles and techniques would be used that would be applicable and effective for defining any large, complex system. Such a task is understandably not a trivial one. There are, however, some common sense "rules of thumb" that can be used to simplify such a task significantly.

In order to successfully define a TSRC and thus build a target system within the context of a TSRC (a large system), TSRC can be treated as a collection of and/or derivable from a common set of small systems. This means finding and gathering an original set of empirical data (which represents a set of "subsystems" and their relationships) in the target system environment, decomposing it from its original form (which is not understandable) and recomposing it into a new form which is understandable. This new form is defined in terms of a common set of mechanisms, themselves systems.

Defining a system with a common set of mechanisms is not unlike the process of using a common subroutine for a computer program. It is rather moving the concept of subroutines to a higher level, i.e., the

requirements definition phase. This process, of course, not only involves the use of common mechanisms but it also involves having the ability to be able to determine candidates for commonality using the requirements.

Any two systems can share a common mechanism in one of two ways. They can either make direct use of the same mechanisms or they can use mechanisms which are derived from the same mechanism . Likewise, derived mechanisms can be shared as well. If one were to carry such a process to its extreme, a given system could be defined with only one mechanism.

To make use of this concept, it is necessary to have a formal system or a consistent set of rules upon which to build the mechanisms. Such a process requires an extra emphasis on the "front end" effort, but it more than makes up for its weight in gold throughout the remainder of the development process as well as throughout its deployment.

An important issue in defining a system is that of determining which objects belong to which system, what the relationships of these objects are, and what the properties of these relationships are. There are certain properties which are inherent or generic to all requirements definitions of systems. All requirements definitions, for example, are made up of data types (objects) functions (which relate members of data types) and structures (which relate functions). If one were to view the acquisition process as a system, a possible data type would be requirements, a possible function would be the process of defining a particular set of requirements and a possible structure would be the process of defining any set of requirements.

Some requirements definition properties appear to be generic, when they are, in fact, not. For example, it might be considered appropriate to make the assumption that all requirements definitions for a large system are potentially interface error prone, and that there is no way to prove if there are interface errors or not. Certainly, no one would argue this fact, today, with respect to the acquisition process itself. Given, however, the use of a technique to define a set of requirements which eliminates that class of errors which are either data or timing conflicts and a means

to demonstrate the correct use of such a technique, one could eliminate the property of not being able to show if there were interface errors or not.

First, then, it is important to understand the properties that systems requirements have, in general, irregardless of the method used to define those requirements. Then it is important to determine those properties in system requirements which result from the technique used to define those requirements. Discussed here are those properties considered to be desirable ones for a given system requirements definition. It goes without saying, of course, that the lack of each such property could be considered as an undesirable property. There is no attempt, here, to distinguish between those properties which are mandatory ones for a successful system definition to have and those properties which would be just nice ones to have, for such a distinction could vary from system to system. That is, an interface error could result in a mission catastrophe in one system, whereas an interface error may only cause an inconvenience in another system.

Many of the properties discussed here are directly or indirectly related to each other. That is, one could automatically cause another one to result. Or another one may not be successful if it does not have still another one to go along with it.

It is our opinion that it is not beyond the current state of the art to be able to define a system in such a way as to have inherent in its definition the properties which are discussed below.

One should keep in mind that the properties, themselves, form a system which itself must be defined with the proper techniques in order that it may have the properties that make it be an effective system.

Fully realize the power of a definition: Any system definition, if it is effective, will have the necessary and sufficient information to convey the meaning of the system to its users and its developers. Not only should it provide the user a means to define his needs and the developer the means to realize these needs, but it should also provide a means to the user to verify that his needs have been realized.

Definition in terms of a control hierarchy: The advantages of defining a system in terms of a hierarchy are well known. Almost all system development methodologies have at least a graphical method for step-wise refinement of a system. It is important to be able to visualize a system definition both with respect to what it does (level by level) and how it does it (layer by layer). A definition in terms of a hierarchy can run the risk of not being reliable, however, unless there are some explicit rules which insure that each decomposition is a valid one. Otherwise, an improper decomposition will continue from that point on being at least just as improper as the previous one (with the exception of one error cancelling the other one out!). That is, there must be some way in a definition of showing that the behavior of successive lower level (or layer) completely replaces the function above it.

Integration of generation and behavior: There are many methods used to define a system. Some of them have to do with generation of output, given a particular input and others have to do with behavior. If, for example, one chose to define a system at the level of a programming language, or at the level of some of the specification techniques [35, 36] that definition would be concerned with generation. If, however, one chose to define a system algebraically (e.g., abstract data types) that system would be concerned with behavior [37]. Others have methods for both, but emphasize that it is a matter of opinion as to which should be used [38]. Often the algebraic methods attempt to define generation [39, 40]. None of these methods in themselves are sufficient for defining large systems. The generation, only, methods provide no means to verify behavior. The behavior, only, methods provide no means to verify generation. The algebraic methods which attempt to define generation become bogged down with a large set of axioms for a non-trivial system. The solution is to integrate a generation method with an algebraic one.

First, the data objects for a particular system definition are determined. These objects are defined in terms of their behavior. A generation technique can make use of these definitions by relating its inputs and outputs to members of these data types in terms of primitive operations on these data types. The result is an integrated system definition both whose behavior is known and which shows how particular members with a given behavior are generated.

Interface consistency: Techniques now exist that can be used to define a system in such a way as to eliminate interface errors. Once a verification is performed to check for the correct use of such techniques a system can be shown not to have these interface errors (i.e., data and timing conflicts).

It is now conceivable that interface problems can be minimized both during an actual "mission" between systems or during the process of developing systems which are sharing common modules or common support tools. The first case is one of real time mission interoperability. The second case is one of interoperability in the development of a set of systems which may or may not be working in the same environment with respect to mission operability. The success of both, however, is directly related to the success of the Army system as a mission.

Of particular interest in this regard is the fact that given the use of interface error prevention techniques, such errors can be eliminated during the requirement definition phase and not at some later phase in the development process, such as in the Full Scale Development or Deployment.

Formal input for automatic tools: A requirements definition which is formal is one which is explicit enough to be understood by people (e.g., combat developer and material developer) and machines the same way. Not only does a formal definition prevent misunderstandings (and thus eventual conflicts) between people, but it also serves as input to automatic tools which are able to automate the process of a system development. A formal definition can serve several other purposes as well. In one Army system we took some existing data and function specification definitions and proceeded to formally define these same objects in terms of data types, functions, and structures. This process not only uncovered several errors in individual specifications and inconsistency errors between specifications, but it also cleared up the meaning for several developers of the particular specifications which were redefined [41]. It has, for example, been determined that if the proper techniques are used to define a system that an automatic analysis process could statically check for interface errors, thus cutting verification costs by an estimated 75% [42]. In this case, this property, i.e., that of being formal, is directly related to one of interface correctness.

Given appropriate techniques, certain tools can now be developed to automate the development process of a target system. In fact, even their conceptualization was not even considered before, since it took such techniques to suggest these tools. By the use of the same techniques, other tools can become unnecessary because the reason for them has been eliminated.

Flexibility for changing requirements: A system must inherently plan for the unplanned and expect the unexpected. These properties are important both when the system is deployed and when it is being developed. In almost all systems, requirements are always changing. Certainly this is especially true in the acquisition process. There are several reasons for this. Mission requirements are always changing due to, for example,

what was learned in a previous deployment of systems. Each new set of users brings about new requirements to meet their collective needs. Better technology, or at least different technology, is introduced as more is learned about what a system can or cannot or should do. Errors found in the system often uncover the need for other requirements. And, sometimes there is always the temptation to either fine tune an algorithm or to make memory or timing a little more efficient. In some cases, no doubt, "better is the enemy of good," but the fact is, each change, more often than not, affects a system definition more than it should, or perhaps in some cases, not enough. This flexibility for change is needed both with respect to changing conditions in a real mission as well as with respect to changing conditions in the developing of such a system. We have found that those solutions which are effective in making a system flexible in a real-time environment are the very same solutions which make the development process of that same system effective.

Sometimes, changes within the environment of a system development process force requirements to change when it should not be necessary, since the requirements were not defined in such a way as to be independent of implementation detail. An example of such a requirements definition is when a set of functions have been defined with specific timing assigned to each function. The only way to have obtained such timing constraints is to have knowledge of a particular machine environment. Problems then arise when a different machine environment is required, since then the original constraints are no longer applicable. But, even in the same machine environment, start and stop times, for example, could vary with a multiprogramming scheduling process when one process of higher priority interrupts another one.

To avoid such a problem, we have found that, whenever possible, the requirements definitions themselves should always have the dynamic properties specified on a relative basis (e.g., as in the acquisition where ASARC is specified to come before DSARC within each life cycle phase [12] rather than ASARC at a specific time and DSARC at another specific time).

There can also be certain "rules of thumb" for flexibility considerations. A multiprocessing environment is more flexible, for example, than a multiprogramming environment. A multiprogramming environment is more flexible than a sequential programming environment. But, one caution: as more flexibility is gained, consider that there must be more control (discussed below) in the use of that flexibility.

Traceability: If a requirements definition does not have a means to explicity trace input access rights or output access rights for every element in a system, that system which has been defined cannot be controlled. Likewise, the same holds true for any change (and its effect) made to that requirements definition. When, however, a requirements definition has the property of traceability, not only is

one able to trace the effect of a particular instantiation of a system, but the dynamic verification of such a system can be cut to a bare minimum, since it is no longer necessary to verify the area of the system where that particular instantiation is known not to travel.

Within the acquisition process itself, this property would ensure that a particular set of inputs was obtained from the proper sources. Thus, for example, certain types of requirements might always be known to come from the material developer. Others might come from the combat developers. Others might come from a higher level (i.e., the integration source).

Ability to error detect and recover: The ability of any system to have the property of detecting errors and recovering from them cannot be overemphasized. Certainly, in a real-time mission it is obvious that such a capability could prevent a mishap or even a catastrophe. But that same feature could be as important for the development of a system. For, if in the development process it is possible to detect an ongoing implementation as costing, say, millions more than another possible implementation, such a realization could be treated as an error and recovered from if go no-go decision points were inserted into that development process. For example, during APOLLO there was a time referred to as "Black Friday," when certain programs were scrapped, even after a lot of time was spent on them since it was considered even more costly to keep developing them. The simple fact was that there were other items that were considered to be of higher priority than those that were in progress. The important consideration was that there was a mechanism available to make such decisions and there was no stigma attached to such a decision. The driving force was the overall goal of the mission. Something, however, had to drive this force. And in order to be effective, that person or mechanism, had to have available the proper input and resources for the decision-making process to restart at every step in the mission or at every step in the development process. This integrating control function must ensure that the alternative route in case of an error is a reasonable one and not one where either the possibility of an alternative "slips through the cracks" or where one or more alternatives work at cross purposes to the original goal. These latter two possibilities were the most common pitfalls that we observed in those error detection and recovery processes that failed.

The Life Cycle Model [12] identifies several go no-go decision points. However, the functions to be performed should a no-go decision be made, are not evident. That is, if it were necessary to try again or terminate a particular target system development process, it is not clear that this could be performed in a graceful and timely manner.

Ability to relate to multiple dialects: Many systems have more than one dialect adhered to by its users. NATO has multiple spoken languages used by its participants. The Air Force, Army and Navy all have their own dialects. Various Army users have their own preferred dialects. The trick is to decide where the cutoff point is in allowing users to

speak on their own terms. Certainly, users from different countries are not always expected to speak the same language. And it would be naive to think that the various services would want to use the same jargon, or even for that fact the different agencies within the Army.

On one of our projects, there was complete flexibility allowed in defining macros for a particular software program. But there was such a diverse use of syntax that no one person could read the program. On the other hand, there are systems (e.g., when one Army system is interoperable with another and both systems can be mostly self contained) where it does not make practical sense to have all the users and/or developers from both systems use identical dialects. (We have noticed, however, as an aside, that it does make a lot of sense for the users and developers of the same part of a system to use the same syntax. More about that later.) In the integration of Space Shuttle flight software requirements, we were forced to contend with several dialects - FORTRAN, MAC, HAL/S, English and equations to name a few. Some were even verbal. In some the level of detail was too much, in others too little.

In order for a system such as the acquisition process or, for that matter, PDSS as a system, to relate to multiple dialects, it must have available a common means to integrate these dialects. Several methods have been attempted to facilitate such a process. One that has often been used is to translate everything to one common language of communication (e.g., in the Shuttle, everything was translated to HAL/S). Another is to attempt to get the various users to use the same equations or, in the case of software, subroutines for the same type of problem solutions. Both of these solutions help, but there is still the problem of misinterpretation of meaning, or semantics. What this suggests is that these systems, in particular the Army systems, could be defined in terms of a common set of semantic primitives.

Defined in terms of semantic primitives: If two or more systems are defined in terms of a common set of semantic primitives, there is always a meeting ground for technical discussions, design sessions, agreements and disagreements. The importance is that whether or not there is an agreement or a disagreement, that agreement or disagreement is well understood. To have a common set of semantic primitives is not only important when there are multiple dialects, for often the case is that users and/or developers may be using the same dialect, or syntax, but each interprets the syntax in a different way. In the software area, developers attempt to communicate in common through a compiler, but often there are developers who use different compilers for the same language. Even if the developers use the same compiler, there is often ambiguity about the way a compiler interprets a given syntax. But, in the requirements world, there is not even the equivalent of a compiler-like vehicle for aiding communication. Once a common set of primitives is established, non-primitive mechanisms can be derived. Likewise other mechanisms can be defined in terms of the non-primitives. In any case, the meaning of these mechanisms can always be traced back to the primitives. Once systems begin to be defined in terms of common

primitives, then more mechanisms can be used as common mechanisms within one or more systems.

In the acquisition process, we noticed, for example, that all phases of the life cycle model had several mechanisms in common between them. And often in these cases where a set of functions in one phase appeared at first not to have a common set in another phase, it was later determined that there was no good reason for such commonality not to exist. In such cases, there was either something missing in one phase or something extra that was unnecessary. Othertimes, different notations or different names for mechanisms within more than one phase obscured commonality that really was there, but which was not obvious until we attempted to understand the meanings of these mechanisms.

Ability to integrate top-down/bottom up: Many discussions have taken place about the pros and cons of the top-down versus bottom-up techniques. In our own experience and the experiences of several others, systems need to have the ability to add requirements from the top or the bottom. The important issue, here, is that all modules in a system should be defined with standard interface rules, so when it comes time to add a new module to a system it won't be necessary to change either the existing system or the new module just because the rules for defining either system were not compatible.

There will be, for example, some modules in a development process that are further along in development than others. There should be no reason why the part of the system that is further along than the other part should sit and wait until the other part is ready. There have been stories of projects where a pure top-down approach was attempted. The result was that a lot of time and money was wasted while developers were waiting to know all top level requirements before they began to work on the next level of development. The fact is that the top level is not really completely known until all levels have been fully defined and the necessary iterations of definition have taken place. Certain components of a system lend themselves to being defined and developed on a parallel basis both at the top and the bottom levels. If you know, for example, that certain types of data will reside within a system, it is possible to define the data types early in the definition process. Certain structures that are known to be ones that can be used in common can also be developed early. Likewise, in a particular mission such as an avionics one, an early phase could be developed to completion while another one was still being conceptualized. Or a compiler for a given computer could be developed before all of its users. One module could be in the full scale development phase while another was still in the conceptual phase. Once the proper standards are established the development of a system can be both multiprocessed and multiprogrammed.

Evolving requirements definition library: Once a requirements definition library exists, it is possible for a system to share common requirements modules. These could be data types, functions or structures. In order to be successful, however, the library not only must be used, but it

must also have inherently several of the properties discussed here in order to serve as an effective tool. For example, given common semantic primitives, mechanisms can be added to the library that are defined in terms of mechanisms already in the library. Or standard interface rules throughout both the library and the systems using the library allow for a library to evolve in an orderly manner. Such a process represents, in essence, the equivalent of an extendable requirements definition language, which in turn can serve as management standards for the development of systems using the language.

Not only is there less to define for a given system if modules already defined in the library are used, but also there is less to verify. Modules in the library are already verified. Thus a developer performing a verification on a system using the library needs to verify only that part of the system not using the library.

There are many candidates for library mechanisms in the life cycle model. Examples of structures and functions are the ASARC and DSARC processes and the OT and DT processes which exist in every life cycle phase. Examples of data types are requirements and specifications which also exist in every phase. The later phases are just at a lower level of detail than the earlier phases.

Easy to understand: A given requirements definition should be as simple as possible and be easy to understand for both a user and a developer. A requirements definition should also be easy to understand for higher level users and developers as well as lower level users and developers. The ability to understand a system definition is strongly dependent on the syntactical methods used to define a system and the resulting syntax from the use of those syntactical methods. For example, if users and/or developers use a familiar syntax in defining a system, they would have have an easier time understanding their own definitions, at least initially, than if they used a syntax that was not familiar to them. And even if it is familiar, the particular syntax they choose influences the understandability of a system definition. In the software world, for example, a FORTRAN program is usually considered easier to understand than an assembly language program. In the requirements definition world, a requirements definition language is considered by some to be easier to understand than English, for example. Others prefer just the opposite. The ideal solution is to have the ability to express a requirements definition in a language as close to a natural language as possible, but with the rigor of a formal requirements definition language. And since a given system or set of systems might have users and developers with more than one natural language, such a feat is only possible if the system allows for the introduction of requirements definitions with multiple dialects. Thus, there is a close relationship between the property of being easy to understand and that of allowing for multiple dialects. Tangential to being easy to understand, then, are the attributes of being able to use dialects that are familiar as well as the ability of having the freedom to express. Again, if such a system is to be successful, the property of having a common set of primitive semantics strongly influences the property of being easy to

understand. For there must be a common meeting ground within which to integrate the various common dialects.

The Life Cycle System Management Model depicts by graphics (c.f. Fig. C-1 of [12]) the relationship of what is to be done, in what order, and by what mechanisms. Such a model is more effective in communicating some very complex concepts than most we have had experience with on other systems.

Transferability: The ability of being able to transfer one system to another environment is indeed a desirable one if a system is going to be around for awhile. It is also desirable if more than one set of users will eventually want to use the system. Such is certainly the case with the PDSS environment. Transferability issues appear and reappear within several layers of a system development. The most common cases occur at the front end or the back end of a system. That is, either a new set of users will need to learn how to understand a system or a system that was running in one machine environment is required to run in another machine environment. Most systems are not able to handle either type of transfer without some difficulty. This is because a system is usually not designed with plans for transferability before the fact. Once techniques, however, are taken advantage of that can define systems to be transferable at either end, those same techniques can be used to define systems in such a way as to be transferable at other points of a system development process. Consider, for example, a system which was prepared to use an operating system which is independent of a particular computer environment. If it then became necessary to transfer this system to another computer environment, that system could be moved intact, as required, to run that system. The scheduling mechanisms, for example, would not have to be redone for a new machine environment. On the other hand, it may be desirable to have a more efficient operating system which is independent of a computer environment. In this case, that same system minus the operating system could be transferred.

Structure integrity: A requirements definition should be an instance of the behavior of the users' needs. That is, the requirements definition has the same behavior as the user model with additional constraints of its own. Conversely, the user model should be an abstraction of the requirements definition. A specification should be an instance of the behavior of a requirements definition. An implementation should be an instance of the behavior of a specification. Unfortunately, however, this is not usually the case. A usual mode of operation is to forget about the requirements (or at least forget about keeping them current) once the specification phase is begun or to forget about the specification once the implementation phase is begun. And the specification and implementation of a system are usually verified on a self-contained basis with occasional ad-hoc checks back to the previous phase. Suffice it to say, the evolvement from one phase to the next is usually not a traceable one. If, however, the results of one phase were treated as an instance with respect to behavior of the results of the previous

phase, such an instance would have to work just as an instance would work in the performance pass of an automated system. To do so would automatically require a carry over of the previous phase, since you could not "run" it without it. To liken this situation to the acquisition process the validation phase would produce a system which is an instance of the conceptual phase, the full scale development phase would produce a system which is an instance of the validation phase, etc. (Figure 4.3-1).

That is each development layer in a given phase can be viewed as a definition which is implemented on a system of constraints to create the next development layer:

REQUIREMENTS = User Model
on User Model Constraints

SPECIFICATION = Requirements
on Requirements Constraints

IMPLEMENTATION = Specification
on Specification Constraints

IMPLEMENTATION = User Model
on User Model Constraints
on Requirements Constraints
on Specification Constraints

Thus, by partitioning a system with respect to its successive development layers, where each layer maintains the structure integrity of the previous one, not only is the previous layer definition intact (or requirements, since all of these terms are relative), but also a given system can be selectively transferred at any point in its development, from one environment to the other. That is, a different user may want to use only the user model of another user, or only the specification, or he may want to use the system completely implemented. But at least he can make a choice as to the cut-off point. Or, the same user may find a better way to specify or to implement. He, too, is able to make a choice as to the cut-off point.

Requirements definition completion criteria: A requirements definition should have explicitly defined criteria for end points. Any given requirements definition is not logically complete until such criteria are met. A very common problem with defining requirements is that there is no way to tell when you are done or if the requirements handed to you are done. There is either too little or too much information. Either dilemma can result in a costly system development process. Various projects attempt to solve this problem in quite informal ways. Engineers are often told to stop their definition process when they hit level n of a hierarchy. Others are told to stop when they hit the "level of implementation detail." Still others are told to define a set of requirements to the level of detail consistent with the information

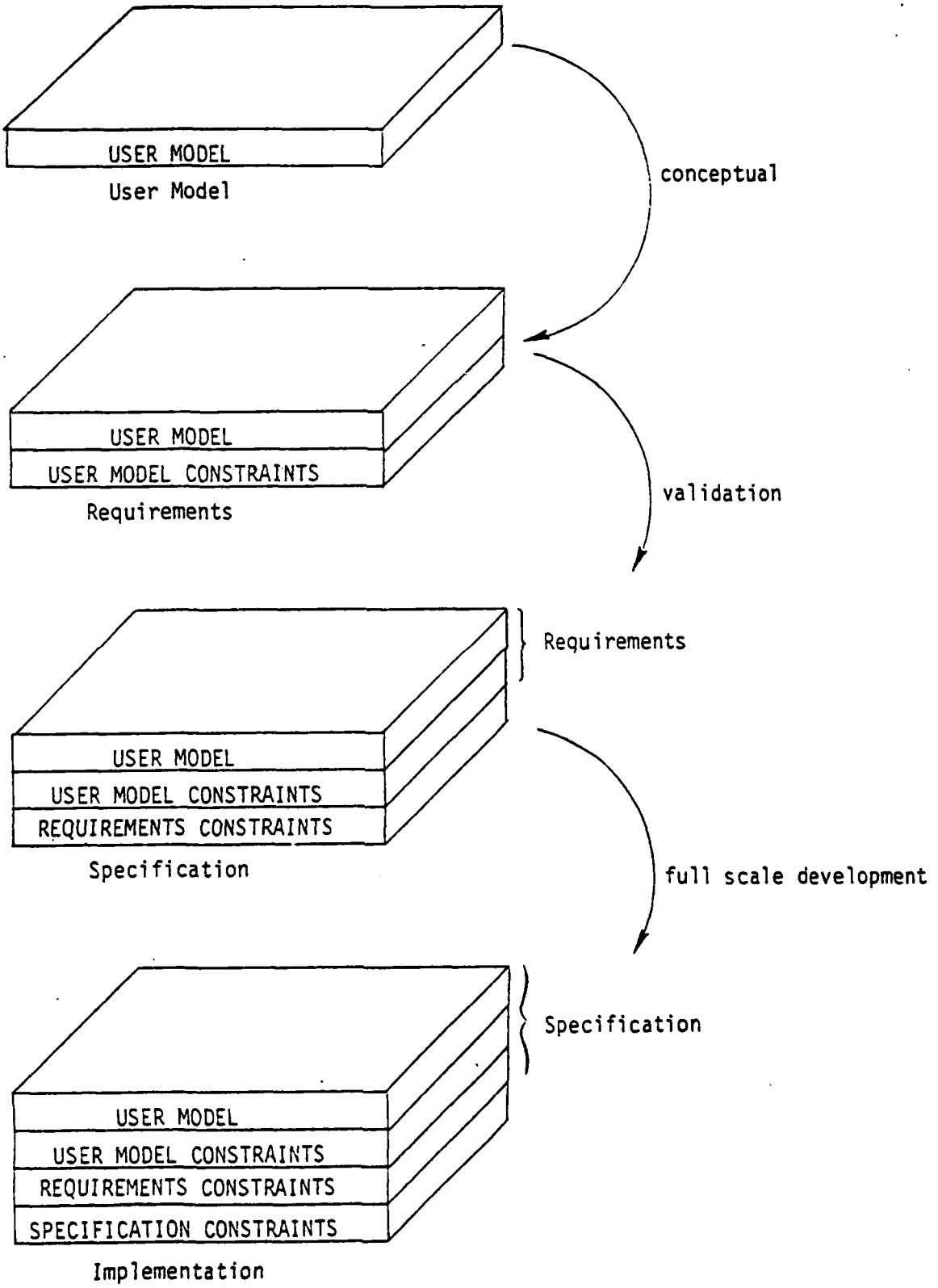


Fig. 4.3-1 Maintaining Structure Integrity in the Development of Target System Layers

they have. These methods have serious shortcomings, because there is no way to measure completion. One way that we have found to be a successful one is to first define the data types for a given system. Each data type has associated with it a set of primitive operations. Once all the bottom nodes in a given hierarchical definition have been defined to the level of detail of primitive operations on data types, the system definition is said to be logically complete. Such a technique shows one where subsystem definitions are incomplete. It also prevents one from thinking a definition is done, when it is not, or from pouring over irrelevant information when the definition is too detailed. In addition, completion criteria help to define check points for each review process.

Common communication vehicle for user and developer: The most successful requirements definition projects take place in an environment where both the user and the developer work on and understand the requirements together. This type of working relationship accelerates the understanding of the user requirements by the developer, and for that matter, the user himself. But, also, such a process uncovers misunderstandings between the user and the developer. It also finds errors before development begins. When a user and a developer work out the requirements definition early in the game, there is often a knowledge on the part of the developer that the user doesn't have, that if the user did have it, he would not have defined some of the requirements the way he did. For example, the mere change of a specified accuracy could save "millions" during the development process and yet not affect the success of a mission. If a developer is knowledgeable about such requirements he might be able to identify a cost trade-off requirement with respect to possible computer environments that the user may not be aware of. The result of such an interchange could be a new iteration in the definition of the requirements.

Given that the relationship between the user and the developer is one of feedback, a common means of communication for such an interchange should be provided. Certainly, many users do not want to converse in a programming language which may be a natural one for the developer to converse in. Similarly, the developer may feel that the language of the user is not explicit enough for him to decipher the requirements. The solution ties in directly with some properties discussed above (i.e., the ability to have a common set of semantic primitives, an evolving library of mechanisms and the ability to handle multiple dialects). In the case of the user/developer interchange, however, it would seem most effective to have the user and the developer work together on either the use of library mechanisms or defining new more appropriate ones. If they both help to define the mechanisms needed and the accompanying syntax, the system, so defined, will "belong" to both of them.

Requirements reside within TSRC: The requirements definitions of a target system should not be considered ready for development unless they have been defined and verified within the context of a TSRC. A system can be defined as a self-contained system, but if the users can't relate to it and it won't work within the users' environment, then what good is it?

In order, therefore, to understand a set of target system requirements it is necessary to understand the user environment of the target system as a system including the users of the target system as a system.

A system can be defined as a self-contained system, but if the developer can't develop it due to lack of technology or resources, then what good is it? In order, therefore, to develop a target system, it becomes necessary to understand the integration of the user and the developer as a system, as well as the integration process of the requirements definition and the development of the system reflecting those requiremnts as a system.

Traditionally, a target system is verified with respect to its relationships after the fact, unfortunately, during full scale development, or even later. What ends up happening is that the job that should have been done at the requirements definition stage is performed by users or support teams in the field during deployment or post deployment: the PDSS system, among others, is forced to play the role that users and developers (including those of PDSS) should have played in early phases. The end result is a much more costly development of the target system since often much of the target system has to be redone, after the knowledge of the real users finally comes into play. Why not bring that knowledge into the system at the front end, i.e., during the requirements definition phase itself? This front end emphasis would conceivably iron out most of the problems during the conceptual phase with, perhaps a few others to be worked out at the validation phase. A bare minimum of problems should remain toward the end of the development process. Ideally, of course, all problems should be resolved before development. And the point is that every problem found earlier saves dollars.

Requirements reside within Army TSRC, Dod TSRC: A most cost effective approach is to define each new Army system within an environment of an overall integration of Army systems (and eventually with an overall integration of DoD systems). It is impossible to really understand thoroughly an Army system until we understand the Army system. The Army system is an integration of all systems, including their relationships, within the Army.

The military has accomplished an integration of organizational structure in some arenas, e.g., a well-understood hierarchy or chain of command; Procedures, uniforms, etc. are standardized for each level of control; Certain laws, customs, etc. are upheld with respect to these hierarchies.

Unfortunately there is not a similar mode of operation in the development of Army systems. There is an attempt at standardization, but even when standards exist, there is no method of ensuring that they are followed. This fact is, no doubt, a result of a proliferation of requirements and technology within an extremely fast moving environment, with no established control mechanisms to keep system developments in check. Thus, the development of systems is not really in control. Languages are not standardized. Techniques are not standardized. Tools

are not standardized. Interfaces are not standardized. Criteria for go no-go decisions are not standardized; in fact in many cases they are non-existent. Goals are not well defined; or when they are, they are often known to be inconsistent with each other. Worse yet, there are several goals or systems that work at cross purposes without anyone knowing about it.

The only solution is to attempt to understand and integrate the Army systems within the Army as an Army TSRC. Once the nature of the Army system is understood, standardization of concepts, languages, techniques, tools, interfaces, and all the rest can begin. Approaching the integration of the Army system means both the integration of the Army system as an eventually deployed system and the integration of the system which develops that set of systems which is to be deployed. Such a system is the acquisition process itself.

Goal driven requirements definition: A requirements definition without well-defined goals is essentially useless. For, without well-defined goals, one does not know if a system is defined in such a way as to perform its mission. Even if the goals, as initially defined, are incorrrect, there is something to start with. There is also a fighting chance to determine if the goals of one system are consistent with the goals of another system.

Throughout this requirements definition process, a concept of goal should always be present. Even at the very beginning of a system definition process, goals, as fuzzy as they may be, help to focus the overall effort. As more knowledge is gained about the system, the goals may be determined to be incorrect, and a reiteration of top level systems requirements may be a necessary step. But the knowledge of even knowing that such an iteration is necessary at the front end can again save dollars and time.

Integrated viewpoints: Sometimes in attempting to be modular, system developers divide a system definition in an artificial manner. As a result, a well intentioned modularization process can end up being more error prone than if it hadn't taken place at all. For example, such a phenomenon has taken place in many higher order language definitions at the software level of a system development process. Priority hierarchies are often treated apart from data scope hierarchies. And, often there is no way to determine if the two hierarchies are consistent with respect to each other since they are not projections from the same system. [43].

If a system requirements definition really "does its job," it should inherently couple those objects which should be coupled and allow for the decoupling of those objects that lend themselves to a natural separation. For example, a function can be naturally decomposed into a set of of lower level functions. But the data flow of these functions should be directly related to the order in which these functions are performed. A decoupling of these notions could conceivably end up in having a function which should receive data from another function being defined to be performed before that other function.

The more familiar viewpoints in a system development process are those of the manager, designer, implementer, verifier, and documenter. Their respective disciplines are management, design, etc. It is easy to see that sometimes a manager designs or verifies or sometimes a designer manages. And it is also clear that such viewpoints are all relative. But given a point in time when we view, say, a person as a manager and he is viewed as managing, or a person as a designer and he is viewed as designing, both disciplines have some elements with respect to the development process in common. For example, they both must have a means to analyze a portion of a system definition. Too often, each of the disciplines is purposely given a different version to work with, since managers, for example, are considered different than designers. And also, too often, each of the disciplines is working with a different revision of the system definition, usually not on purpose.

There is no reason why the different disciplines in a system development process need their own specified form of a system definition. True, they may be interested at different times in different modules or different levels of detail in the system. But this is also true with a set of designers.

If a system definition is not sufficient for one discipline, it is not sufficient for another one. A proliferation of different versions merely adds more paper work to deal with as well as compounds the interface problems between everyone concerned.

There are times, however, when one group of people would like to see a different description of a system than another group or a projection of a description of a system. But, before such a description is produced the definition being described must be understood and consistent. Then it is possible for several different viewpoints or different forms of one viewpoint to be produced. We have found that if a technique is used which produces a consistent definition first, one can safely translate to another description form which may be more familiar to a different group of users. For example, we have started with an AXES definition [44] and translated the description of the definition to other forms such as IDEF [45] and PASCAL [46]. Figure 4.3-2 shows the types of projections that can be automated, given that the intended systems definition has logically consistent and logically complete information to start with. The reverse process, however, is not possible unless the missing pieces or inconsistencies are resolved first. We have also been able to start with an AXES definition and then produce projections such as a precedence flow, data flow, dynamic representation, priority hierarchy, data hierarchy, etc. Obviously, one could not do the reverse process.

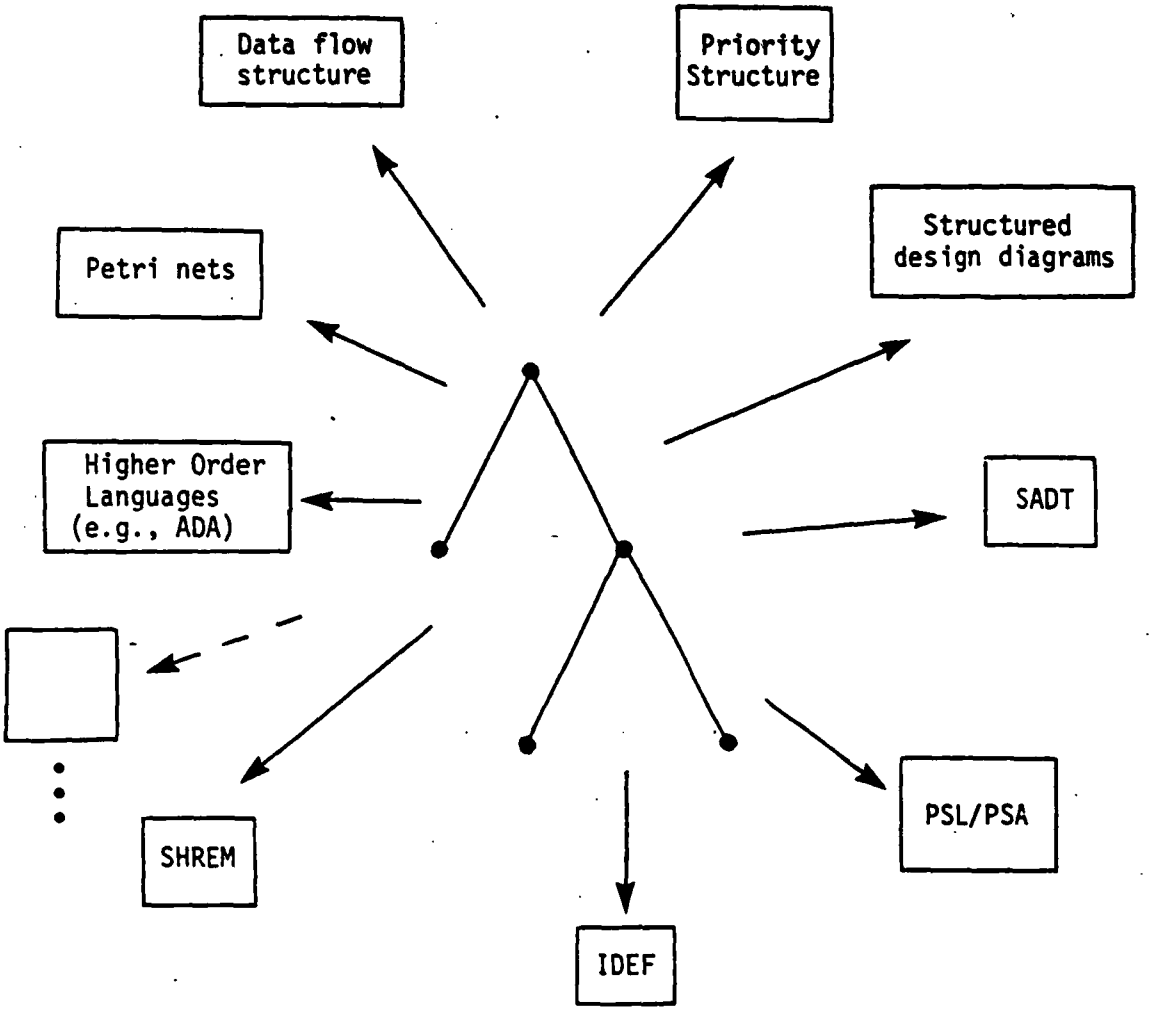


Fig. 4.3-2 Examples of Possible HOS Projections

4.4 How to Make Use of Properties in a System Development

If a system has properties such as discussed in Section 4.3 developers can capitalize on these properties in developing their own systems. To name a few, we have made use of these properties for designing user requirements [47], [48], and in [49] and specifications [50], [41], [51], and [34]. In [41] we made use of the same properties for verification. Other efforts have made use of these properties for implementation [51], [46], and others for documentation [52], [45], and [53]. A discussion of how we made use of such properties for various functions such as learning, specification, management, etc. in Army systems can be found in [41]. Included in this section, as an example, are "high-level" checklists which could be used for the management, design, and verification processes. Similarly, checklists for implementation and documentation can be developed.

MANAGEMENT CHECKLIST

The manager is concerned with control from several systematic viewpoints. These include the target system development process, including the system of people who develop a system and the target system itself. In each case, there are several aspects of control that must be considered. They are control of:

- which functions or processes are to be invoked
- proper relationships between inputs and outputs
- access to output data
- access to input data
- error detection and recovery
- ordering of functions

The manager, in order to be effective, should have a means to guarantee that the systems he is responsible for have methods which concern themselves with these aspects of control. These methods also must be under control in that they should be consistent with respect to each other. Appendix V is an example of a more detailed checklist which is a "how" to the above checklist as a "what."

Although more formally we define management as control [54], the human manager* must also often be the equivalent of a resource allocation tool (RAT [54]) who is both responsible for and often manually serves as an implementator tool. We therefore use the ideal system properties to support the definition of a system from a manager point of view both in terms of who is in control and in terms of who makes and receives certain assignments [55]. The second main feature of a manager is analogous, of course, to what goes on in a compiler-like environment when assigning certain programs to a computer.

DESIGN CHECKLIST

With or without a methodology, a designer, as a type of person, is not ever going to be like any other designer. That is, no two designers are going to use a method in the same way when it comes to how they use their own creative processes, i.e., how they acquire insight. But the proper method can accelerate a creative process by prompting thoughts that may never have occurred without it. Table 4.4-1 shows what the designer should consider in designing his system, keeping in mind that the desirable system would have the set of properties discussed in Section 4.3 inherently in its definition. Keep in mind that this is an iterative process and that the steps are not necessarily in the same order for any two design processes. An example of the use of this type of process is shown in [34].

---

*Note: We use the concept of manager, here, in the sense of being able to perform a certain set of "chain of command" functions that could be performed by more than one individual. Eventually such processes could be automated. Here the function that is to be performed takes priority over who performs it. That is, the concept of "manager" is not to be confused with the concept of "leader." We identify a leader with such properties as charisma, compassion, power, etc. It is not possible now for us to define a leader formally. Maybe it never will be. In the case of a leader, 'who it is' that performs such a function takes priority over 'what it is' that is to be performed. It is possible, however, for a person who is a manager to also be a leader and vice versa.

Table 4.4-1

Design Process Checklist

- Organize the system hierarchy in terms of the three basic units: data types, functions, and structures.
- Find already existing types of units (mechanisms) in the library (i.e., data types, functions, and structures) that can be used in the system definition.
- Determine if more abstract data types should be defined in order that the system hierarchical definition can be completed at a higher level of definition, leaving the lower levels to be defined in terms of behavior rather than procedurally.
- Determine if more abstract mechanisms can be formed and/or derived from existing mechanisms.
- Create new types of mechanisms that are not in the library.
- Look for commonalities within the new mechanisms. Get rid of the redundant ones.
- Add new mechanisms to the library.
- Continue a system hierarchical definition until all functions at the bottom of the hierarchy are primitive operations of the chosen data types.

VERIFICATION AND VALIDATION CHECKLIST

By verification we mean checking the logical completeness and consistency of a system. Validation is the term used for checking the correctness of the intent of the system. Within the concept of a TSRC, verification can be performed automatically and statically with respect to the TSRC description. Validation requires instantiations of the target system to be checked.

The verification and validation checklist (Table 4.4-2) capitalizes on the use of the properties discussed in Section 4.3. The intent is that the use of such a checklist would cut the cost of verification significantly. These steps are to be performed in the order that they occur in the table for each module that is to be verified.

Again, all of these testing modules can also be submitted to a library so that the one time only philosophy, as well as the other steps of verification, can be applied here as well, i.e., in the testing of the tests. With an approach, such as the one discussed above, one need only dynamically verify that part of the design which is concerned solely with the performance of a particular algorithm with respect to the particular original designers intent.

Table 4.4-2

Verification and Validation Process Checklist

- One time only - Define a system using as many modules in the library as possible, since these modules have already been verified. Only their application needs to be verified when they are used in a newly defined system.

- Conceptual phase of verification - Define a system using techniques which eliminate the need for certain types of verification. If, for example, techniques are adhered to which eliminate data and timing conflicts, it is not necessary to look for these types of errors. Thus, for example, wire tracing tools, in such a system environment, are obsolete.

- Static phase - Verify without running or simulating a system for all those areas which can be checked for the proper use of techniques in the design of a system. If the system is designed using the proper techniques, such a static verification can be performed automatically.

- Dynamic phase - Here testing itself should be viewed as a system:

  - define the interoperable environment of the target system as a system

  - define the structures of testing scenarios as individual systems

  - define instantiations of these scenarios as systems.

SECTION 5.0

WHEN TO DEVELOP SYSTEMS WITH DESIRABLE PROPERTIES

5.0 When to Develop Systems with Desirable Properties

Since it is not beyond the state of the art to be able to define a system with the properties discussed in Section 4, is it, from a practical standpoint, a reasonable approach to be taken now or in the near future?

Incorporating these properties in systems already deployed or far down the line in the development process is not an easy job. In some cases, in fact, it is almost impossible. There then becomes the unenviable choice of either fixing existing systems or redoing them completely over again. Often, it is much more cost-effective in the long run to start over, although such a fact is not obvious until it is too late in that much time and money is wasted in finding this out by attempting to fix a system first. This is often true when requirements for a particular system are in a state of flux. For those systems which have more or less stabilized, or for those systems which are near obsolescence, a complete redo may not be such a wise choice. But for those systems which are near the front end in their development or which are changing almost as often as to behave as a new system, there appears to be no reason why the use of newer techniques would not be a major consideration. If eventually all new systems had applied to them the same standard techniques, then, among the obvious benefits, there would no longer be that unenviable choice of deciding on an ad hoc basis which systems had which techniques applied to them.

With those systems where a decision has been made not to change to new techniques there are still some benefits that can be obtained from the new techniques. An attempt can be made to take an ongoing specification defined by conventional methods and convert that existing specification to a parallel prototype one using the new techniques. Such an effort is very effective as a front end verification tool, since many errors can be uncovered in the translation process. Not only does this process find errors, but it finds errors without running the system dynamically and it finds them early. The new converted specification can also be used as a means of understanding the original specification if it is still considered desirable to keep it intact as the primary system. And, eventually if the primary system has had so many changes as to be un-

wieldy, there is always the back up system to transfer over to should such a transfer be considered necessary. We have found such an exercise can be useful for verification and validation purposes even when a specification has been implemented within a particular machine environment. In this case, several errors can be detected that could be fixed both in the original specification and in its resulting code.

Suppose a choice is made to start from scratch and use the new technique. A very common problem in a system development process, especially when there are a lot of people involved, is that everyone is reluctant to change to different techniques, even if they are aware that if they had used them the development process would have been much smoother and the life cycle would have been more cost effective. Reasons for this are many. But the main reason seems to be one of familiarity in the use of the techniques. Thus training and public relations must take a key role when a new technique is introduced. But these are not easy tasks. A technique is not going to work unless people know how to use it and once having done so others are able to interpret the results of its use in performing the various disciplines such as design, implementation, and verification.

We, therefore, have to concentrate more on techniques of transfering knowledge on a system wide basis. If the technique used allows for multiple choice of dialects, then the designers of new mechanisms for defining a system could work together with both users and developers at defining a chosen set of syntax. A more conservative approach is to stick with the dialect already used by developers, but define it in terms of the semantics of the new approach. Still another approach might be defining the system with the new approach, but then translating that definition to the language of the developer, so that the syntax would still be familiar.

Ultimately we want to be able to define systems, whether they be hardware, software, or humanware, or some combination, by merely collecting modules from a library(Figure A I-1 Appendix I). There would only be the choice of what should be done and how it should be done. Once such a choice is made, the "what" and the how can be collected from the library of system modules.

SECTION 6.0

A PRELIMINARY ANALYSIS OF

THE

LIFE CYCLE SYSTEM MANAGEMENT MODEL

6.0 A PRELIMINARY ANALYSIS OF THE LIFE CYCLE SYSTEM MANAGEMENT MODEL

6.1 Preliminaries

With some preliminary analysis as background, the next step was to attempt to understand more explicitly the requirements definition phase (or conceptual phase) of the Life Cycle System Management Model. A rough control map of the conceptual phase was sketched out, using as background information, the detailed description in [12]. Several questions arose during this preliminary exercise, some of which were answered in later stages of our effort and some of which are still not answered. For example:

1. Why aren't in-line requirements obtained from, for example, logisticians and trainers just like they are from HQDA, combat developers and material developers?

2. Conversely, if there is to be made a distinction between in-line efforts and support efforts, why is there not a support effort for HQDA, combat developers and material developers? For example, material developers may have a simulation to support the verification of the target system. Combat developers and material developers may need a requirements definition language ir order to support the communication which is necessary to take place between them. HQDA may require a management system which could track milestones of the target system development.

3. And where is PDSS in the Life Cycle Model Conceptual phase? It would appear that PDSS should be helping to define requirements at the front end just as does everyone else who has a vested interest in the target system. And PDSS, as does everyelse, has its own set of support requirements.

4. There appears to be a missing area completely in the Life Cycle Model. This area is the common block or library which HQDA, operational tester, combat developer, material developer, logistician, trainer and PDSS use in common. Such a concept is not only cost effective in that common mechanisms are used, but it also can serve as an integrating function which prevents redundant or inconsistent development efforts.

5. There were no explicitly defined no-go checkpoints. For example, if several task forces or members of an ASARC or DSARC should disapprove of a set of requirements, what happens? Are there official mechanisms for redoing a set of requirements or for terminating a system development completely?

6. How are paralled efforts developed? That is, when one module is finished too soon, or one module is finished too late, which

is more often the case, how are all of the various modules in the system coordinated with respect to timing, resources, and interfaces?

7. Who or what mechanism controls or integrates the process of a phase? For example, in cases where both a combat developer and a material developer are responsible who arbitrates a disagreement and finally overrides an opponent, when there is one?

8. Sometimes it is hard to distinguish in the life cycle model between input and functions. For example, the material concept investigation results in a letter or agreement, but what is the output of, say, the HQDA approval or is HQDA approval an output of something such as a meeting of the HQDA or a special study by one or more of its members?

9. Which processes are functional in nature and which processes relate functions? For example, one process such as the HQDA could result in just the special task force meeting. Another could result in both the special task force and the special study group meeting. Such a mechanism could be represented by a structure. However, the special task force itself may always be able to be represented by a function, unless upon close examination such a mechanism could also vary in a functional sense.

10. How are all the forms of requirements either integrated at one step or maintained between steps? For example, does a letter of agreement get carried through as far as DSARC, or does just part of it get carried through, or is it replaced by later documents?

11. What happens, and where, and how in the conceptual phase when an error or change in requirements is found in a later phase, e.g., full-scale development? Who is responsible for introducing and accepting such a change?

12. Testing is emphasized in the support area of acquisition, but such discipline, in our opinion, is very much in line. Not only should it be performed or accounted for in some way at every new step of definition, but the way a system is defined affects the way it is tested or not tested. Dynamic testing is performed by subjecting a target system to its environment. Knowledge of that same environment is needed to define the system. If the definition of the environment is needed in the in-line effort for the definition of the target system, it should be included in the in-line effort for the definition of the verification of the target system. If, however, one is considered a support effort, the other should as well.

Once a preliminary control map was constructed for the conceptual phase, we attempted to get an idea of what the control map would be like for other phases. For example, how did they differ or where were they the same as the conceptual phase. Interestingly enough, on first glance the similarities far outweighed the differences. And where there were differences, either something was missing in the conceptual phase or something was missing in the phase which was different from the conceptual phase. In other cases functions or processes which appeared at first to be different really, in fact weren't once the behavior of those processes was analyzed. And, in still other cases, a slightly different ordering of processes would mislead one to think that they were different, when they weren't. Some examples of questions that arose during the analyses of other phases are:

1. Why are operational testing and development testing functions begun in the validation phase and not the conceptual phase? That is, verification should begin with the front end requirements.

2. Isn't there also a possible award of contract at the beginning of the concept phase? Conversely, isn't it possible that there is none in other phases?

3. Why do the identification of long lead time requirements occur for the first time in the full scale development phase?

4. Who is in charge of coordinating each phase? Of all the phases?

5. Which processes are performed iteratively? Which in parallel? Again, is there a no-go milestone before a new phase has started?

6. Why is there a separate entity assigned for testing the combat development requirements (i.e., operational tester) and not the material requirements?

7. Why is hardware singled out and not software?

8. Again, why does the PDSS involvement come even later, i.e., not until the end?

6.2 The Army Life Cycle System

The intent of this section is to identify the system of Acquisition, the system of Post Development, and the relationship between Post Development and a subsystem of Acquisition, Concept Formulation within the environment of the Army Life Cycle.

As stated in "Major System Acquisition," Circular No. A-109[56] the,

> System Acquisition Process means the sequence of acquisition activities starting from the agency's reconciliation of its mission needs, within its capabilities, priorities and resources, and extending through the introduction of a system into operational use or the otherwise successful achievement of program objectives.

Post Deployment is defined in Appendix A of the PDSS Management plan [31] as the

> cost effective support of deployed Army Defense systems.

In order to understand what Acquisition and Post Deployment are, as systems, it is useful to define the environment in which these systems reside, i.e., the Life Cycle system. According to Circular No. A-109 [56], Life Cycle is inferred from the definition of life cycle cost to be,

> the design, development, production, operation, maintenance and support of a major system over its anticipated useful life span.

Such a system has many interactive components, feedback between subsystems and within subsystems, asychronous communicating subsystems (many of them intended to operate in parallel), critical decision points, functions allocated to people, machines and software, etc. In the sense of Miller [57], the Life Cycle system has the characteristics of a "living system." To define any system there are basic units and diagram conventions that are used in this section. These units and diagrams are described in Appendix III.

6.2.1 Environment Of Acquisition And Post Deployment - The Life Cycle

Inputs to the Life Cycle (Figure 6.2-1) are the available products and technology that the Army has access to from previous acquisitions; the new needs that initiate the analysis that may lead to a new acquisition and the time for initiating the Life Cycle. The goal, or output, of the Life Cycle is to produce the final product, within the cost and schedule constraints, and to maintain the information from proposed changes to the intended product so that any new acquisition can learn from previous acquisitions. Each of these inputs and outputs of the Life Cycle is associated with a particular time. That is, in Figure 6.2.1, each "x" associated with a subscript. i, (where i is "p" or "R" or "R_0" or "R_0") is a variable whose value is a State (of Commodities) (See Appendix IV for data type definitions). Each t_i is a variable whose value is a Time. A Commodity is something which can be bought or sold, it can appreciate or depreciate, it can be a collection of other commodotites, and it has some value at a particular point in time. A State associates something of a particular type with time (e.g., a State (of Commodities) has two components: one is Commodity and the other is a Time). We want to be able to talk about Commdities and Time and States (of Commodities) because in defining the Life Cycle we are concerned about cost and schedule considerations with respect to a particular item, be it a requirements document or a piece of hardware.

The Life Cycle is the decider, or integrating function, that controls the relationships among the subsystems of Post Deployment and New Requirements (Figure 6.2-2). Those subsystem levels of the Life Cycle that are controlled to interact asynchronously are indicated in Figure 6.2-2 by the symbol "Ⓐ" (e.g., Post Deployment and New Requirements form a level controlled by Life Cycle to interact asynchronously.) Each subsystem on such a level can communicate and/or perform in parallel with the other subsystem on the same level. Completion criteria are indicated in Figure 6.2-2 by the symbol "||" followed by the name of a predicate function (such as "Obsolete?") which when evaluated to the value True will complete the goal of the controller. Schedule criteria are indicated in Figure 6.2-2 by the symbol "@" followed by the

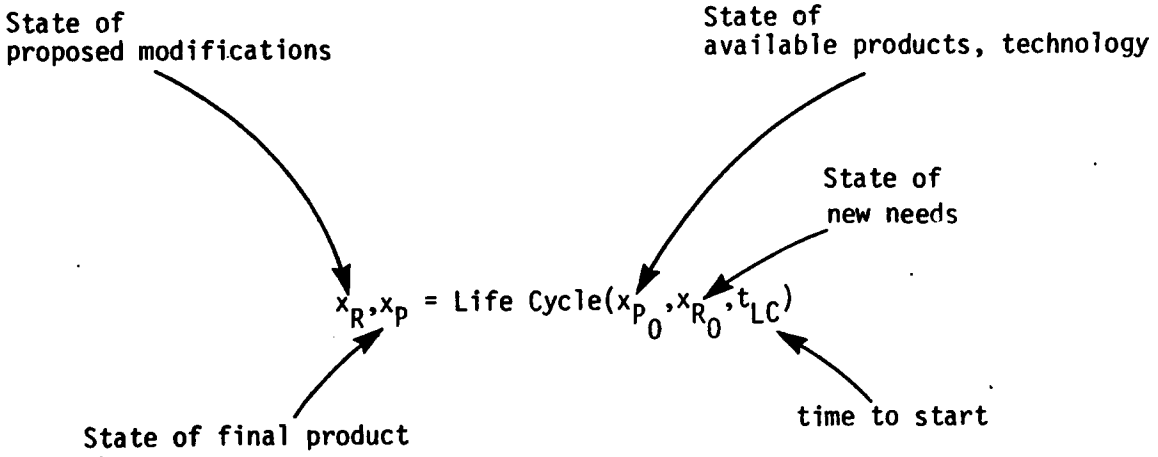


Fig. 6.2-1 Inputs And Outputs Of Life Cycle

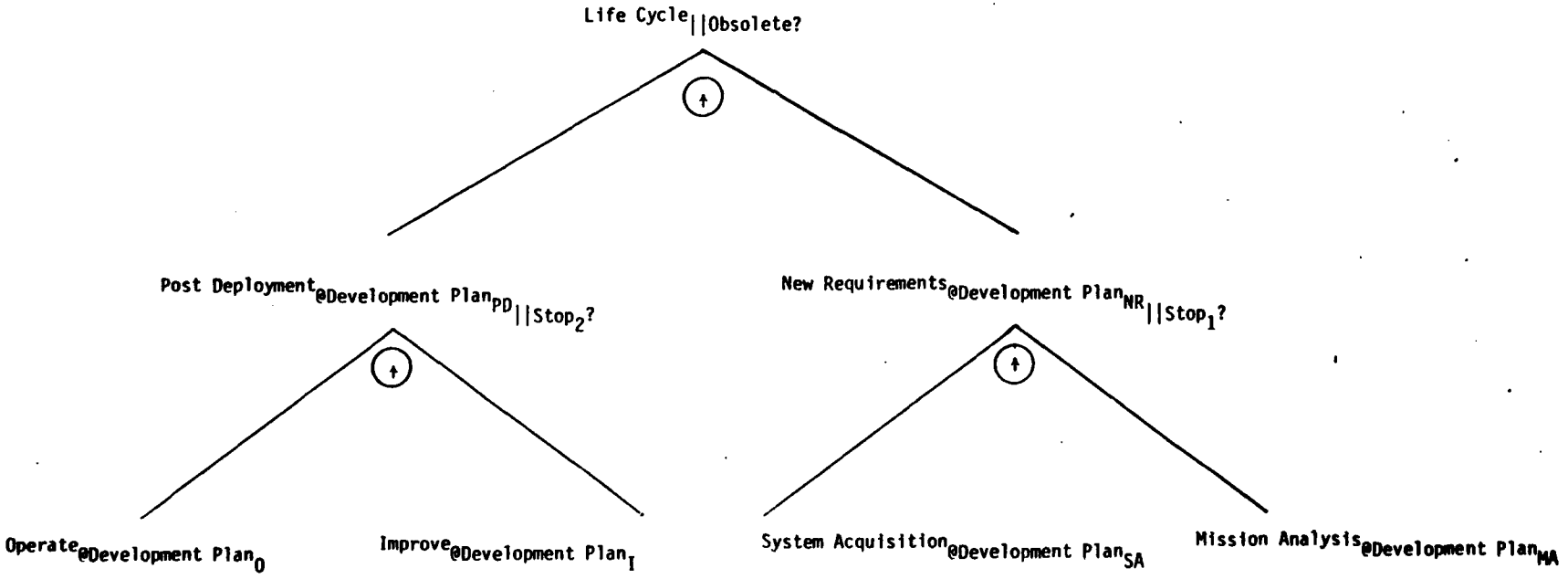


Fig. 6.2-2 The Environment Of System Acquisition

name of a function which uses the most current state information to schedule the next instance of an asynchronous subsystem. (e.g., Development $Plan_{NR}$ schedules the next instance of New Requirements.) The corresponding control structure defining how asynchronous subsystems are functionally related can be found in Appendix IV.

The goal of the New Requirements effort

$$x_p = \text{New Requirements } (x_{po}, x_{Ro}, t_{NR})$$

is to transform the needs, making use of the knowledge of available products and guided by the results of the Development Plan, into the new product (consequently fulfilling part of the Life Cycle goal). The New Requirements effort controls the inter-communication between Mission Analysis and System Acquisition in order to obtain its goal. The commodity of x_p has two components such that

$$\text{Correspondent } (x_p) = \text{Lot } (\text{Correspondent } (x_{p_p}), \text{ Correspondent } (x_{p_R})).$$

The Correspondent operation on a value of State (of Commodity) produces the Commodity component of that State. The operation Lot gives us the capability to combine commodities to form a larger commodity (see Appendix IV).

The commodity of x_{p_R} is the output of Mission Analysis where

$$x_{p_R} = \text{Mission Analysis } (x_{R_0}, x_{p_0}, t_{MA})$$

The commodity of x_{p_p} is the output of System Acquisition where

$$x_{p_p} = \text{System Acquisition } (x_{p_0}, t_{SA})$$

The results of Mission Analysis, when forwarded to Post Deployment, informs the Post Deployment effort of which changes were acceptable and which were not acceptable. The result of System Acquisition when forwarded to Post Deployment is a new version of the initial product.

The goal of the Post Deployment effort is to make use of that product as its set of requirements and to produce proposed changes to the initial product.

$$x_R = \text{Post Deployment } (x_{R_0}, x_{P_0}, t_{p_0})$$

This goal is obtained by controlling the inter-communication between the functions to Operate the current product and to Improve the efficiency and effectiveness of that current product.

The form of the functions Operate and Improve are similar to the form for the subsystems of New Requirements, where the commodity of x_R has two components: one the output of Operate and the other the output of Improve.

The criterion for completion of the Life Cycle is indicated in Figure 6.2-2 by the function Obsolete. From the " (↑) " definition, function Obsolete? is evaluated before each new instance of New Requirements or Post Deployment is initiated. This indicates that a standard criterion for go, no-go decisions is required and specifies which events trigger that evaluation. Both New Requirements and Post Deployment have their own stopping criteria, $Stop_1$? and $Stop_2$? respectively. Within each instance of New Requirements there can be many asynchronous interactions between Mission Analysis and System Acquisition. Likewise, many instances of Improve and Operate can take place to comprise an instance of Post Deployment. Instances of New Requirements and Post Deployment interact asynchronously until the product being developed is considered to be obsolete. When Obsolete? is True, then an instance of Life Cycle is complete. If we were to show the system of developing the Army system there would be many Life Cycle instances demonstrated.

Scheduling criteria within the Life Cycle are indicated in Figure 6.2-2 by each Development Plan function. The total development plan for the Life Cycle can be obtained by defining each Development Plan function. Development Plans are continuously updated based on current information. Cost can be obtained from the operation, Value, which is applied to a particular commodity at a given time to produce money (see Appendix IV). The operation value can be used to determine cost criteria.

Many authors have expressed that it is difficult to graphically represent Life Cycle relationships among subsystems [12, 58]. In each case, an attempt has been made to try to represent each asynchronous interaction while, at the same time, trying to represent the functions that interact in

this way. The separation of structure (Appendix IV) and function making use of the structure (Figure 6.2-2) helps to identify the level of interaction, key decision points, flow of information and subsystem relationships.

6.2.2 Environment Of "Front-End" Requirements - System Acquisition

As new needs are evaluated by Mission Analysis, the resulting objectives are inputs, or requirements, to System Acquisition. The new needs that are generated by Mission Analysis may be the result of errors detected during Post Deployment or the results of a previous Mission Analysis instance. Although each System Acquisition instance asynchronously interacts with Mission Analysis, the subsystems of System Acquisition are related such that the outputs of one phase are inputs to the next phase (Figure 6.2-3) i.e., the outputs of Concept Formulation are inputs to Advanced Development; the outputs of Advanced Development are inputs to Engineering Development; the outputs of Engineering development are inputs to Production Items. Figure 6.2-3 shows the iterations that can occur among and between phases. For example, if one decision is made to approve the results of Advanced Development, Engineering Development in initiated. However, if the results of Advanced Development are not approved Formulation is initiated which, in turn, reiterates the Concept Formulation phase. At each decision point, the decision is either to proceed to the next phase, to regress to the proceeding phase, or to initiate a "no go" decision. Thus, for example, the iteration of Production and Deployment may cycle back to Concept Formulation and then cycle forward to Production and Deployment. The main difference between the relationships of the subsystems of System Acquisition in Figure 6.2-3 and the most immediate subsystem relationships of the Life Cycle in Figure 6.2-2 is that each subsystem output of System Acquisition is an event that initiates a <u>different</u> sybsystem. Figure 6.2-3 shows the synchronous relationships among phases. Section 6.2.3 shows asynchronous relationships within a phase.

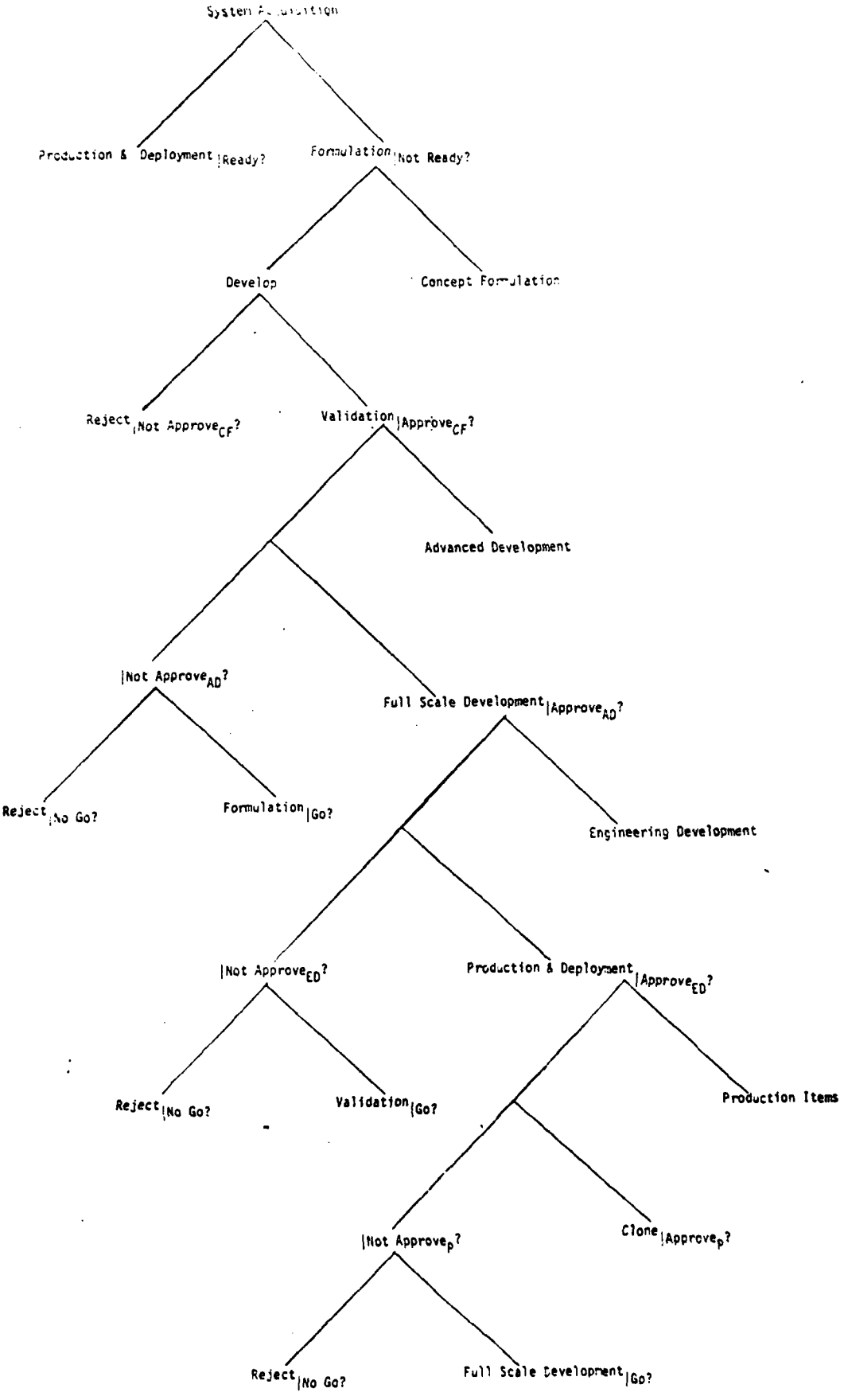


Fig. 6.2-3 The Subsystem Of System Acquisition

6.2.3 The Subsystem Of Concept Formulation

Concept Formulation (CF) integrates the formulation of the Material Proposal and the Operational Concept. The goal is to produce a set of requirements for the Develop Subsystem of Figure 6.2-3 based on a set of specified needs. The structure of Concept Formalization (Figure 6.2-4) has been derived from the functional relationships described in [12]. The agency or command responsible for carrying out a particular function is indicated by the subscripts associated with a particular function name. For example, HQDA is responsible for CF; the material developer (MD) is responsible for the Material Proposal; and the combat developer (CD) is responsible for the Operational Concept.

The Material Proposal uses the most recent information produced by itself and the Operational Concept subsystem. Likewise, the Operational Concept Subsystem uses the most recent information produced by itself and the Material Proposal subsystem. Thus, the Combat Developer and Material Developer make each other aware of the results of their own particular activities. The two subsystems asynchronously interact with each other, often performing their own functions in parallel.

The formulation of the Material Proposal, coordinated by the material developer, is itself an asynchronous set of activities between the material developer and the logistician. The combat developer coordinates the integration of trainer and operational tester activities with activities that he himself is responsible for. The subsystems of the Operational Concept also interact asynchronously.

Key decision points are shown by each $Stop_i$? (where i is "CF" pr "MP" of "OC" or "OT") in Figure 6.2-4. We suspect from a preliminary analysis of the Life Cycle Model [12] that the structures of Advanced Development and Engineering Development subsystems are much like the structure of Concept Formulation.

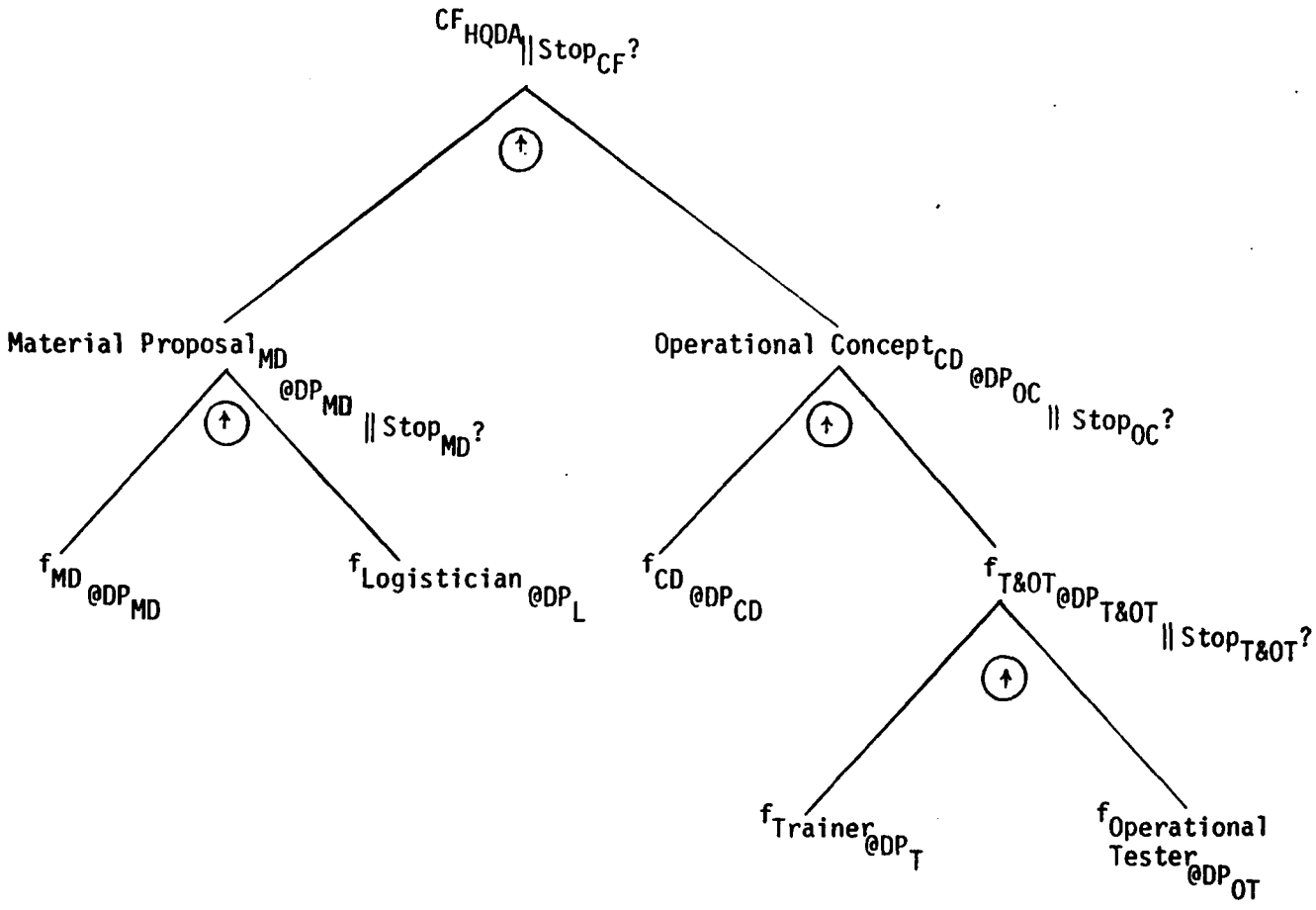


Fig. 6.2-4 The Subsystem Of Concept Formulation

6.2.4 The Subsystem Of Post Deployment

In defining the subsystem of Post Deployment, the distinction between the activity or function of Post Deployment and the other activities that can take place concurrently within the Life Cycle to support the activity of Post Deployment is emphasized (cf Figure 6.2-2).

The structure of the Post Deployment subsystem (Figure 6.2-5) has been derived from [31]. The goal or output of Post Deployment is a set of proposed changes to the initial product. The proposed changes are the result of operating the initial product (subsystem Operate) and attempting to improve the efficiency or effectiveness of the initial product within the constraints of the product requirements (subsystem Improve). In the event that an error is detected during the Operational Procedures or System Improvement Procedures, error recovery is attempted by the subsystem, Latent Error Detection. Relationships between the structure of Post Deployment of Figure 6.2-5 and procedures within the PDSS Management Plan[31] are shown in Figures 6.2-6 and 6.2-7.

6.2.5 The Relationship Between Post Deployment And Concept Formulation

As proposed product changes are produced by Post Deployment activities they are forwarded to New Requirements (Figure 6.2-8). Here, they are analyzed by the Mission Analysis subsystem and desirable requirements are then forwarded to System Acquisition. The System Acquisition subsystem provides the Concept Formulation subsystem access to these desirable requirements. The outputs of Concept Formulation trigger the Develop subsystem which leads to a revised product, completing the goal of Formulation, and in turn System Acquisition, and in turn New Requirements. Post Deployment receives, as input, the revised product from New Requirements.

In essence, once Post Deployment has been initiated, the New Requirements activity can be viewed as a support activity to Post Deployment. The first instance of New Requirements can be viewed as an "initialization" process of the Life Cycle. With this view of the Life Cycle, the same procedures

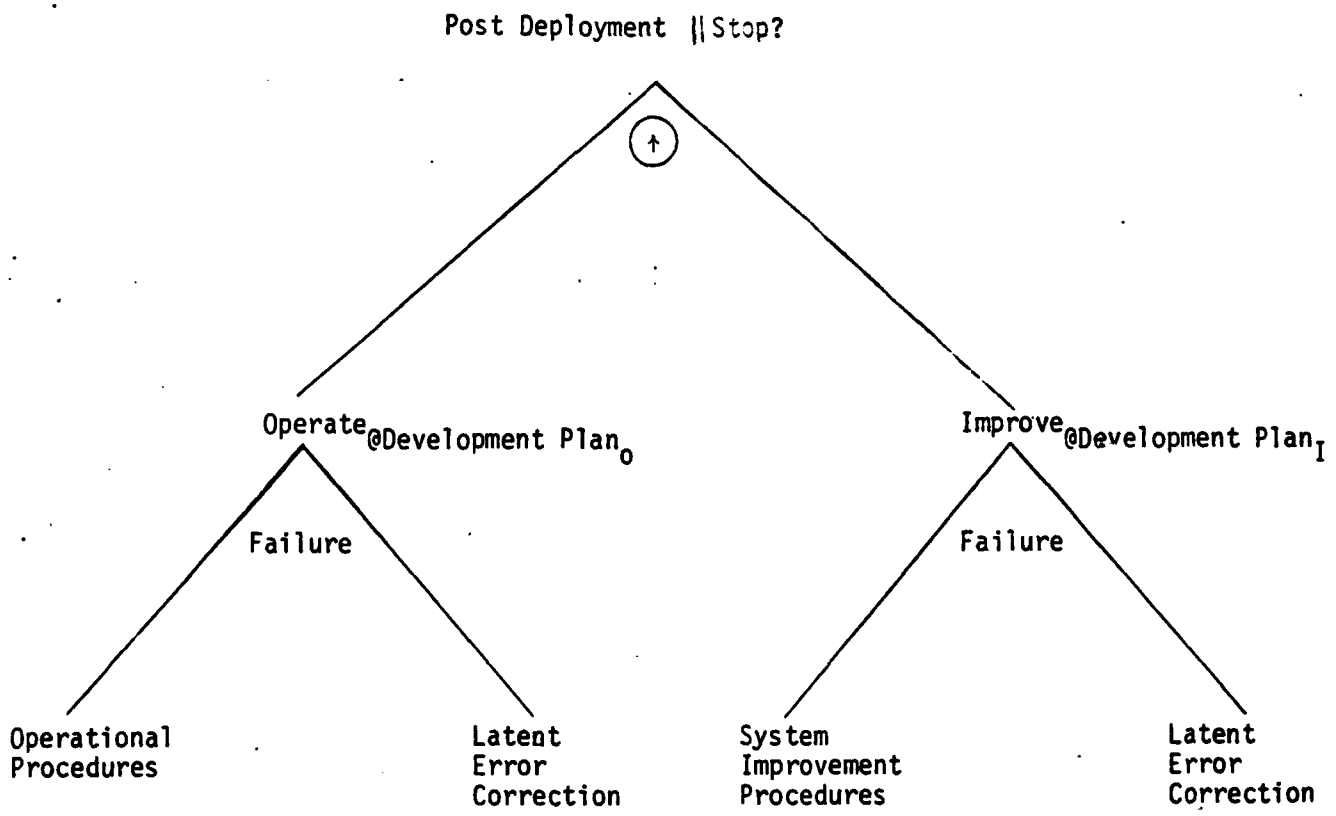


Fig. 6.2-5 The Subsystem of Post-Deployment

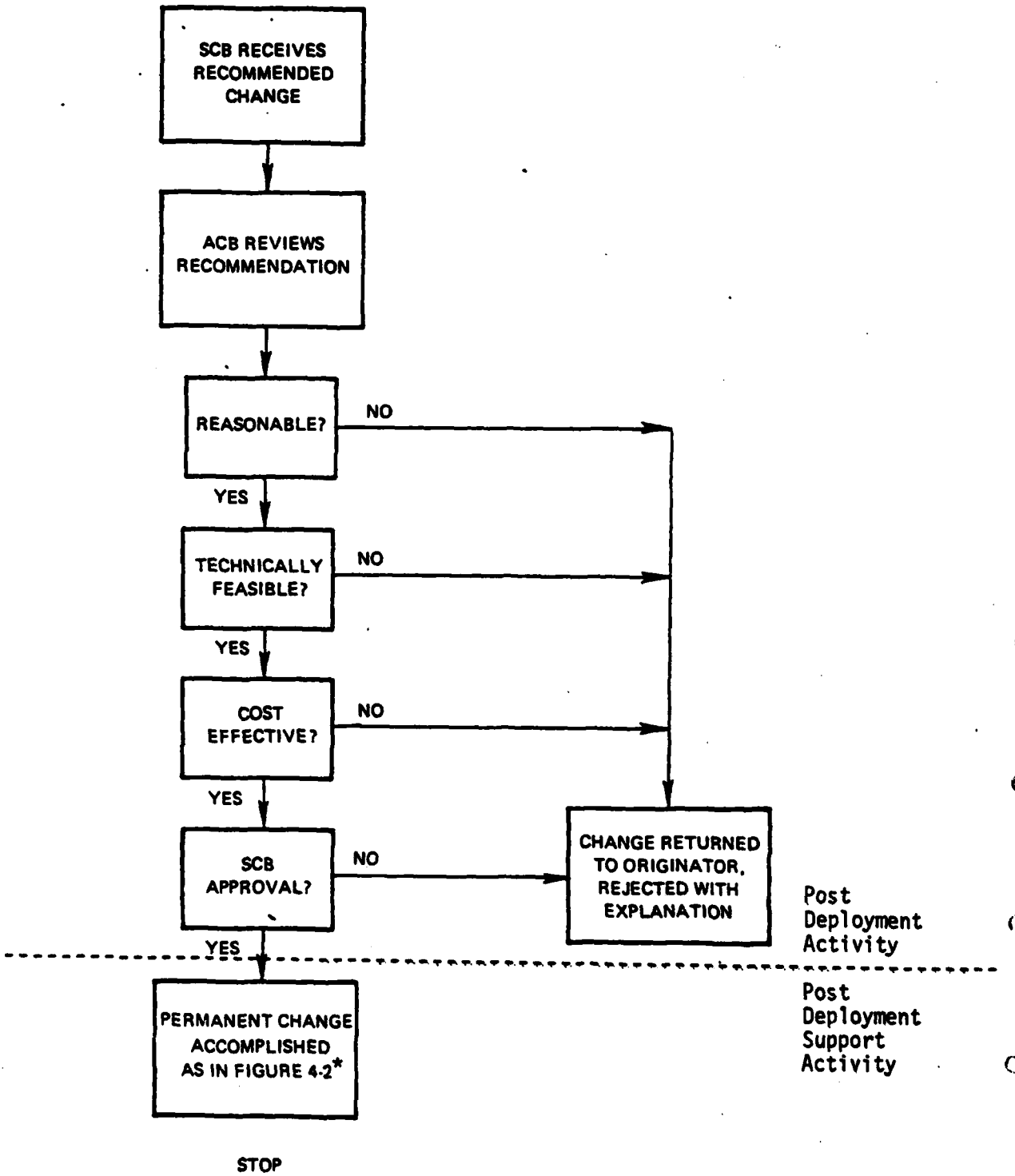


Fig. 6.2-6 System Improvement Procedures
(Taken From Figure 4-4, p.4-19 of PDSS Management Plan [31])

* Figure 4-2 of PDSS Management Plan

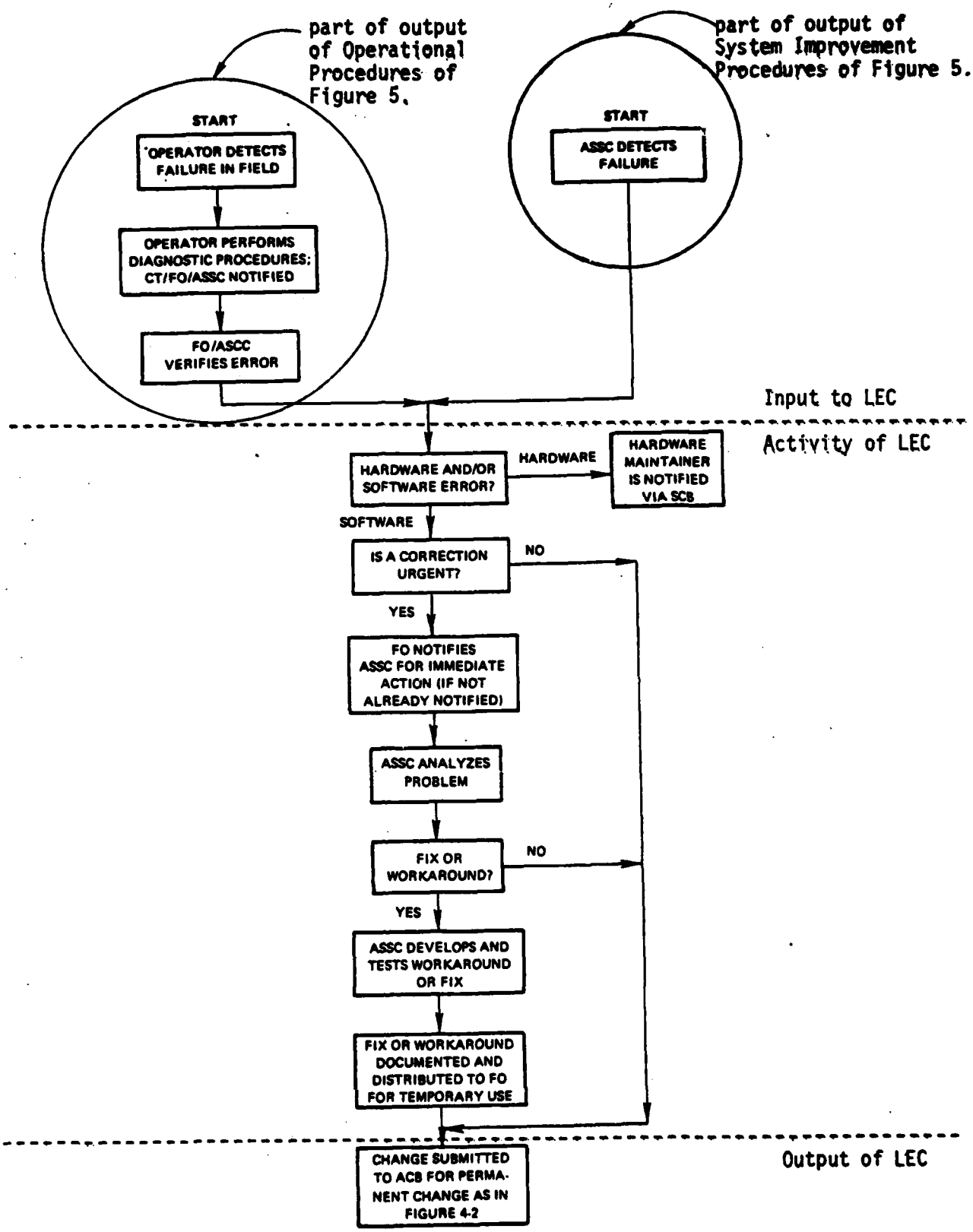


Fig. 6.2-7 Latent Error Correlation (LEC)
(Taken from Figure 4-3, p.4-18 of PDSS Management Plan [31])

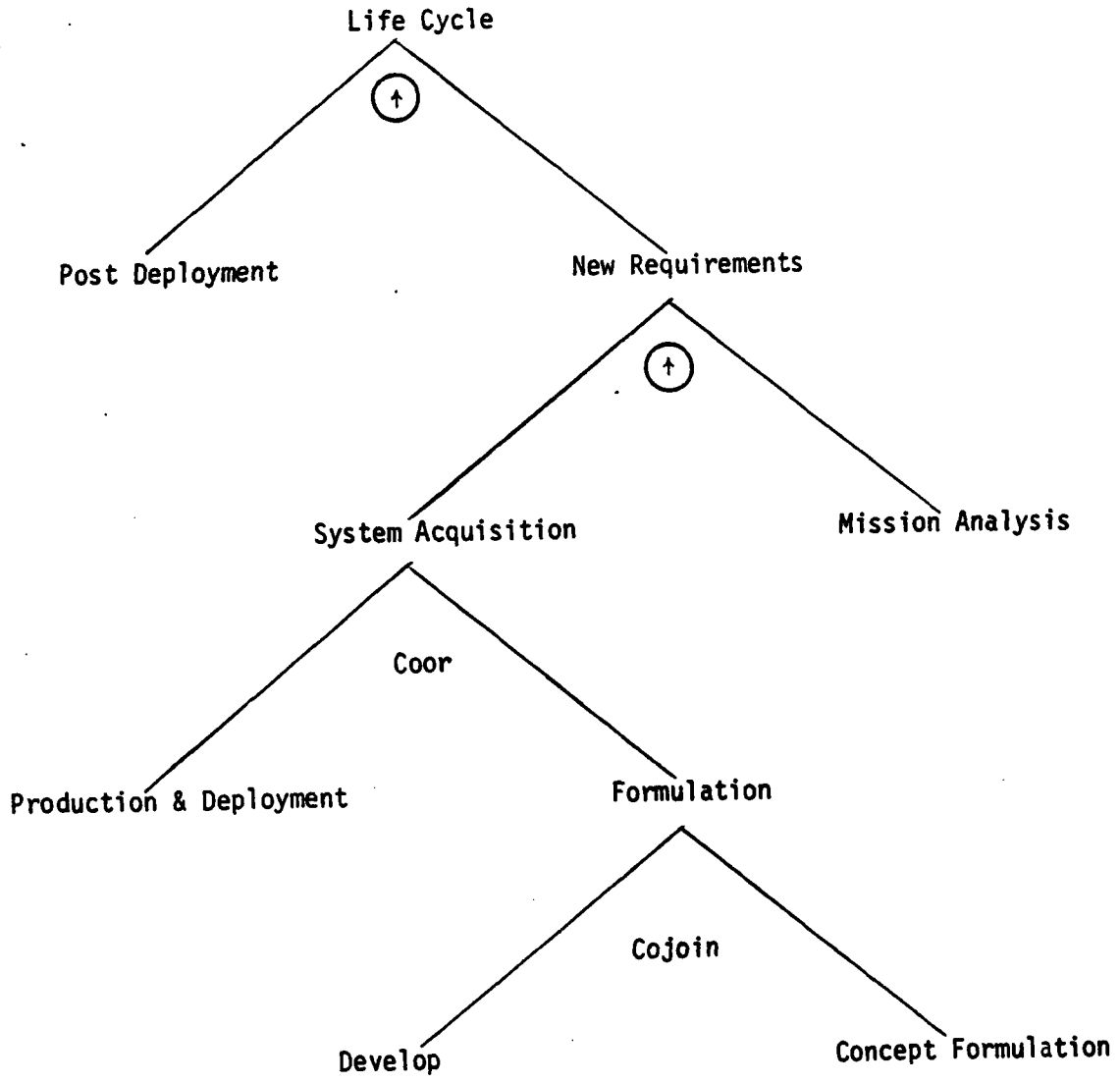


Fig. 6.2-8 The Relationship Between Post Deployment and Concept Formulation

are followed for producing the initial product and for supporting each proposed change to that product. Likewise, the Post Deployment effort can be viewed as a support activity to New Requirements. New Requirements procedures and PDSS software change procedures, as presented in the PDSS Management Plan [31] are consistent with this point of view.

If the product being developed follows recommendations of Section 4.0 of this report, some subsystems of that product can be completed earlier than others. As a subset of the product completes, Post Deployment activities are initiated. Thus, Post Deployment and Concept Formulation are continuously an integral part of the entire Life Cycle.

SECTION 7.0

RECOMMENDATIONS

7.0 RECOMMENDATIONS

These recommendations for the Army acquisition process emphasize front end requirements definition techniques. Although the Army directed us to concentrate on techniques which would impact PDSS, it is clear after analyzing PDSS and its environment that the recommendations would be valid no matter which target system the Army selected. However, the choice of concentration on PDSS was a wise one, since the lack of proper techniques in the "front end" clearly affects the degree to which PDSS is successful in a major way. For PDSS is the area that the use of proper techniques, or lack of proper techniques, would be most visible. And, since PDSS now is shown to be active only in the last major phase of the acquisition process, the result of improper techniques, still not rectified by the time of PDSS, is a much more costly development process for any given Army system and, in fact, the Army system as a whole. Not only are problems (and their effects) in a system maintained throughout its development process, but once that system is deployed there are multiple versions of such a system being maintained. It is clear that the resultant process of debugging multiple versions of a system has to be more costly than the elimination of problem areas in one version of that system before it is deployed, especially when the maintenance of these systems is neither controlled or standardized by a single source.

SINGLE SOURCE CONTROL

In order for any system development and maintenance process to be effective, there should be a single source that is responsible for controlling a system and all of its multiple versions. Such a control in any large system would necessarily be broken down into lower level hierarchical points of control, where each point of control is in

turn a single source of control for the subsystem in its domain. Such a control hierarchy does not imply in any way the need for any particular geographical distribution to effect such a plan. In fact, such a plan can be implemented within one geographic area or several as long as the efforts under each control point are successfully maintaining a "chain of command". This means that these systems are both properly integrated and standardized with respect to their position in the level of hierarchy. This single source control point also does not imply a single person or organization being in control of a particular level of hierarchy. For example, each controller in a chain of command could be made up of a developer representative, a user representative, and a representative who was considered objective enough to break a tie, if there was one. Such a representative could be appointed by others on a particular control team to serve in such a capacity.

OPTIMUM USE OF CONTROLLED COMMON LIBRARY

There is no need to continuously reinvent wheels. The development of Army systems could be made much more effective if these systems could share a common library under a controlling single source. An evolving library could consist of data types, functions and structures. This library would be treated as an Army system just as any other system. That is, various agencies from various geographic areas can contribute to and/or use such a facility.

REQUIREMENTS DEFINITION METHODOLOGY

There is a need to select and/or develop a requirements definition methodology for the Army (and eventually, DoD) which, when applied, would result in Army system definitions with the system properties described in Section 4.3. With such a methodology, not only would each system adhering to it have these properties, but if all systems used

this same methodology they would be standardized with respect to each other, both in development and in real time.

REQUIREMENTS DEFINITION LANGUAGE

There is a need for the Army (and eventually, DoD) to select and/or develop a requirements definition language. This language should be based on the requirements definition methodology recommended. Then, a system defined with such a language would have the properties of a system defined with the requirements methodology. Such a concept (i.e., a common language for DoD) is not a new one. ADA, for example, is now being developed for the programming phase of software development. What is seriously lacking, however, is a language at the front end. And, if the language could define systems with properties discussed above, it would have several features ADA does not have. These include the ability to use multiple dialects, the ability to derive more abstract language mechanisms from primitive mechanisms, and the ability to maintain an evolving and selective set of language mechanisms for each particular user. With a requirements definition language such as the one recommended here, systems could be defined formally, including software, at the requirements level and eventually such a definition could be automated to produce the ADA syntax automatically, if such an intermediate level of dialogue were so desired. When a requirements definition language is used in the future, ADA users can be integrated by defining the ADA syntax in terms of the requirements definition language semantic primitives. This technique could also be used with other languages now currently in use, e.g., FORTRAN, TACPOL, COBOL, etc. Ultimately, systems using different languages should be able to be related. But, such an integration is not possible until a choice of a common set of semantics is made. That semantics can and should reside within the Requirements Definition Language.

SYSTEM DEVELOPMENT AUTOMATED TOOLS

Once a requirements definition methodology and a requirements definition language have been selected or developed, there is need to develop automated tools which are both based on the methodology and which are able to communicate with systems defined with the requirements definition language.

There is some flexibility with respect to the type of support tools that can be used in a system development process. For example, various documentation aids (such as a translation of a definition to a familiar set of graphics) or management information aids or simulator aids are not as crucial to the actual output of the development of a system as an in-line development tool. True, if not properly used, or if they gave improper results, they could mislead a system developer to draw a wrong conclusion here and there, but an in-line development tool (for example, an analyzer at the front-end or a compiler in full scale development) if they work improperly, can unquestionably produce the wrong results. Thus, the control of those tools such as an analyzer or a compiler equivalent (at least the front-end of a compiler) or an operating system (again, the front-end) are much more crucial as to their selection or development or standardization since they can affect directly the rightness or wrongness of a system development output as much as the modules which reside in a system itself. We therefore strongly recommend that the Army should select and/or develop a standard set of such tools for all of its developers. Again, such tools would be developed and maintained in a controlled library just as the system modules themselves, are. In fact, the tools and the Army systems would both share the library modules, since the tools themselves are Army systems. The other tools, i.e., the support tools, could be in the library as well, but these could be optional and their use could vary from system to system.

RELATIONSHIP BETWEEN THE PROPERTIES

Further studies should be made into the area of understanding the system

of ideal properties of any given system. There are some properties, for example, that if they exist, then others exist as an extra benefit. Others, in order to be effective, need additional ones to go hand in hand. Understanding such a system will no doubt have great payoffs in the future. For example, which types of properties were crucial for which types of systems might be able to be determined, or which small set of properties to incorporate into a system which makes possible all the rest. We suggest that the ultimate set of ideal properties for a system can be derived from a small set of primitive properties. We do know at this point that, given the set of techniques which we are using, that there are certain inherent properties which exist. Another set of techniques would imply a whole new set of properties. Which properties suggest which techniques?

AN INTEGRATED PLAN FOR SYSTEM DEVELOPMENT

As a starting point, it would indeed make sense that the Army select a qualified working group made up of informed user and developer experts from various Army agencies to develop an integrated plan for developing systems. Such a plan would include considerations of the recommendations we have made above as well as others that the working group might also recommend. Such a working group should start off as a relatively small one in order that some real concrete requirements could be defined. Once these were completed, a larger working group could be formulated in order to bring in a forum to critique the original working group recommendations. It is our opinion that the PDSS working group is an example of the type of collective experience and talent that would be quite effective in providing this type of forum.

MANAGEMENT INNOVATION PLAN

One of the problems mentioned several times by the PDSS group as being more or less unique to a PDSS type of environment was that of not being able to keep talented people on a so-called maintenance job. Yet when problems arise in a deployed system, it is absolutely necessary that the people who detect errors, understand them, and fix them, are the talented, capable, experienced, and creative people. But the problem is that during the time when there are no challenging problems to worry about, these people get bored. And, even when the challenging times occur, there is always the problem of the system begin developed by someone else. Thus, not only is the engineer involved often without enough to do; but when there is something to do, the part of the system he is working on is not his own. The result is that the good and talented people are always leaving. This happens with both contractors and civil servants. This dilemma, coupled with the fact that military personnel change jobs every two to three years as a normal course of events leaves many systems in a sorry state. It's a wonder, in fact, that they work at all, when they arrive at the PDSS stage of development. Possible long-term solutions are as follows:

1. Move the PDSS personnel involvement right into the front-end requirements definition area. Certainly the system needs their requirements. They'd eventually get them anyway. And, the PDSS personnel will have a more vested interest. The system is also theirs. After all, in the end it should be anyway. And, keep that involvement going throughout the development process. If this is not the case, PDSS personnel will end up defining the requirements for the system on the next iteration. And, if it's non-interested personnel defining those requirements, there are the well-known ramifications that could compromise a system.

2. Use the proper techniques from the beginning of the system development process (see above recommendations). When this happens, there won't be a need for personnel to spend a lot of time finding and fixing errors that should have been taken care of earlier in the development.

Given that from both political standpoints and practical standpoints, the long-term solution is some time in the future, there are some shorter term solutions that appear to be feasible ones, although even these have

their political and practical hurdles to overcome, since the environment is a large and complex one and involves many people and many organizations:

1. Involve PDSS personnel in research and development projects to help alleviate PDSS problems in the future (for example, the type of efforts that the above recommendations could lead to). We have found in our experience that some of the best researchers are the very people who have been burned by the very problems that need solving. Without this sort of experience, some of the research may never even get started because others, normally involved in research only, are not often even aware of what problems need solving unless someone brings it to their attention. And, often that is not enough, because just hearing about a problem does not bring home an understanding of it. Sometimes a problem has to be lived through in order to know what it really is.

   Research and development projects such as those discussed above could either be worked on directly by the PDSS maintenance personnel or indirectly by the PDSS personnel monitoring contractors involved in such projects. Another benefit of such an approach is that the results of such research will be taken more seriously by others involved in related areas, such as other PDSS efforts, since the others will know that the people responsible for the research have an understanding of their own problems.

2. Involve PDSS personnel in the definition of requirements which will eventually result in steps to improve the PDSS environment as a whole. Such personnel, for example, would be invaluable in a workshop(s) such as was recommended above.

3. Involve PDSS personnel in the development of system library modules for Army systems as a whole. There is no doubt that there is a lot of good software within various PDSS facilities that goes by unnoticed because of lack of communication, interest, or coordination between the various Army systems. Various incentive plans could exist for acceptance of system modules into the controlled system library.

These solutions are not recommended as just a means to keep PDSS maintenance personnel happy just for the sake of it. Bored personnel do not keep their minds as sharp as challenged personnel. Again, the result of maintaining bored personnel is a potential compromise on those system(s) which are being maintained.

MANAGEMENT INTEGRATION PLAN

It is strongly recommended that the higher level interfaces within the Army system be formally defined. A start in this direction can be found in [16]. This would involve both the interoperability aspects of the deployed or to be deployed systems and the interoperability of the development processes of the deployed or to be deployed systems. Not only would such an exercise result in understanding these relationships better, but it would also uncover interface problems that exist or could potentially exist. In addition, such an exercise would uncover potential candidates for standardization and commonality not heretofore uncovered. It would not be surprising to see such an effort uncover several candidate solutions for improving the overall cost effectiveness of the continuous development of the Army system.

The PDSS management plan mentioned several plans for organizing the Army system, for example, centralized PDSS by command and centralized PDSS by Battlefield function. There were other alternatives as well [31], Appendix II]. Within this recommended effort of formally defining the interfaces of Army systems or potential Army systems, it would be highly desirable to analyze the trade-offs of some of these alternatives.

TRAINING - THE NEED FOR A STANDARD ARMY DECISION SUPPORT SYSTEM

There appears to be two extremes from which to alleviate the training problems that were mentioned by the members of the PDSS working group. One is to find better methods to train the troops. The other is to find better ways to develop the systems so that the need for thinking in real time or memorizing manual procedures is cut to a bare minimum. The recommendation is that an effort be launched which would prototype a user interface system for a selected Army training environment. Once such a system was shown to be sucessful in its use, it would stand ready as an initial introduction to standardizing and improving user interface systems within PDSS.

Although this particular aspect of the Army has not been analyzed here, we have had experience with various users that had similar problems to contend with. There were times, for example, on APOLLO, when astronauts or the ground control, even though they knew the system they were working with "cold," when the pilots or ground experts were under extreme fatigue or emergency conditions, that there was always the problem of making sure that there was enough time to react or making sure that they would not make errors, which, of course, is human. As a result, several important steps were taken: the user interface was made as simple as possible; the software provided an option of allowing each mission phase to be self-contained and the other option provided for by the software was one which interacted with the astronaut on a go, no-go, key in new data-go basis; an error detection and recovery plan was implemented in the software which monitored for possible human or hardware event driven errors. Once such an error was detected, the recovery system notified the astronaut about the problem, recovered from it immediately by going back to ground zero if catastrophic to continue, or restarted at the last step if not catastrophic. The user was then given several options interactively in order to get back "in sync." A high level astronaut input requirement definition language was used to simulate scenarios. These scenarios were used by the astronaut for training and by the developers for simulating the astronaut. Thus, both the real user and the developer related to the same scenarios for developing the user interface and the same language for defining those scenarios.

It would appear that many of the techniques such as those described above could be used to enhance some of the Army training systems, both to cut down on training time by introducing automated aids and by having less to train due to more automated system approaches at the user interface end of the system.

FURTHER QUESTIONNAIRE ANALYSIS

The questionnaire analysis of the PDSS environment and the acquisition process was extremely helpful both in understanding their environment and in understanding the users of and developers from their environments.

But, we were only able to do a preliminary study due to lack of time and resources. It is our recommendation that these questionnaires should be refined, as a result of various critiques received from the people interviewed, and then distributed to several other members working within the Army community. This exercise would be helpful both to educate ourselves more about the Army acquisition environment and to the Army community to accelerate their thinking about new methods in order to make the development of Army systems become a more effective one. The refined questionnaires should have the questions hierarchically structured in such a way as to allocate various levels in the questionnaire hierarchy to various levels of personnel (i.e., from General to Colonel to the working members on a project).

LIFE CYCLE COST EFFICIENCY TEAM

It is recommended that a Life Cycle Cost Efficiency team be set up, at least within the Army, which is to be composed of members from diverse disciplines and agencies. Such a team would be set up to look for acquisition redundancies. The team could make use of at least the ongoing results of two other efforts discussed above: the effort which defined interfaces within the Army system and its development would uncover redundancies; the effort which set up a controlled library facility would also uncover redundancies. With these results as a basis, the team would know what types of properties to look for in uncovering redundancies. In some cases, redundancies would not be so obvious. The next step would include briefings of the various agencies by the team as well as recommended procedures that various agencies would take to help determine if they were reinventing the wheel. This procedure is not unlike a patent search, once it becomes mechanized.

Throughout the Army, and in fact DoD, there are many redundant research, development, and application efforts. Reasons for this are many. The system is so large and complex that no one person is able to grasp the entire system. The standards for DoD systems are not such that one would be able to spot redundancy without a lot of analysis even if two identical projects were staring him in the face. There is always the "not invented

here philosophy" in any organization including the government. Many systems people are so busy "getting the job done" that they often have no time to communicate to the outside world the ongoing results of their own efforts or to see what others are doing. This is especially true of large complex application efforts. It is not uncommon, for example, to see research papers reporting on new theoretical ideas which were being applied years ago in a real system. The terminology of researchers and application people is very different. Thus, when they attempt to exchange ideas, it is not always obvious when they are talking about a similar concept. New concepts or techniques are often developed for each new application, when, in fact, a particular terminology is often applicable for a family of applications. Even when there is a new technology available and known to be applicable to a particular environment, people are hesitant to spend resources in educating others in the use of that technology. It too requires teaching in order to use it. Often new techniques, since they are new, are considered too difficult to use. As a result, other alternatives are developed which are "easier" to use, but they compromise the original reasons as to why it was needed in the first place. People are often caught up in fads. The government is no different. Yesterday, higher order languages, today specification languages. Tomorrow who knows.

In a current fad, for example, there has been no attempt to define requirements for a specification language. Thus, there is the risk of each customer selecting or developing one with no way to know if it serves his needs. The result is more proliferation of "languages." There is also the temptation to want to automate everything. There must be caution, however, in deciding whether or not a particular automated tool fulfills a necessary function for the customer. In one case, a manual process was replaced by an automated one and the procedures necessary to use the automated tool required more manual preparation than there was without the tool. There can also be the risk of eliminating certain reliable manual procedures for the sole purpose of using something which is automated. (Perhaps, in this case, both might be needed.) Then it becomes necessary for a new set of techniques to be used to make up for the ones that were necessary.

The result of redundancy is not so obvious until the dollars are added up. But when one effort has millions poured into it and another similar one also has millions poured into it, one should ask why. And when several efforts suffer from this problem, it becomes clear that there are areas which could be house cleaned with no negative impact on technology advancements within DoD.

The Life Cycle cost efficiency team would determine if systems are being developed before their requirements are really known and then finalized. Such a function would encourage the go no-go decision right at the front end of a system development. This type of review could result in considerable savings.

One other problem area prevalent in system developments, including the Army, is that one where certain technology is being designed not only by committee but by auditorium. Not only does the fact that there are so many people designing something slow a design process down and also compromise its integrity, but also the dollars for manpower and travel over the country become insurmountable. Several years ago we compared the effort of a design of an operating system by two people to an equivalent one designed by an auditorium of people. The first effort took weeks. The second effort took years. We have found that successful design efforts include no less than two people and no more than three people. This is not to say that a particular problem could not be broken down into smaller pieces, each of which could represent a design effort. But no more than three people should be responsible for a particular area. Another function of the Life Cycle Cost Efficiency team is, therefore, to look for efforts within the Army which are being bogged down by the numbers of people involved in a particular design process. Again, we believe that a considerable amount of dollars could be saved by eliminating designs by "moving auditoriums."

Of course, the Life Cycle Cost Efficiency team itself would not be effective unless it also adhered to a controlled hierarchy in its organization and its own design efforts would want the sort of focus discussed above.

LITERATURE CITED

[1] Hearings on Military Posture and H. R. 12564: Department of Defense Authorization for Appropriations for Fiscal Year 1975, Part 4 of 4 Parts. Committee on Armed Services, House of Representatives, 93rd Congress, 2nd Session, March-April 1974, pp. 3781-3783.

[2] Hearings on Military Posture and H. R. 5068: Department of Defense Authorization for Appropriations for Fiscal Year 1978, Part 3 of 6 Parts, Book 1 of 2 Books. Committee on Armed Services, House of Representatives, 95th Congress, 1st Session, February-March 1977, pp. 73-74.

[3] Refer to [2], pp. 28-30.

[4] Refer to [1], pp. 16-17.

[5] Refer to [2], pp. 2-12.

[6] Salisbury, Alan, B., The Making Of A Weapon System: TACFIRE 1959-1978, National Defense University Research Directorate, Washington, D.C. 20319, August 1978.

[7] Hearings Before the Subcommittee on Federal Spending Practices, Efficiency, and Open Government, Part 2. Committee on Government Operations, United States Senate, 94th Congres, 1st Session, June-July 1975, pp. 95-113, 205-213.

[8] Refer to [2], pp. 296-308.

[9] Hearings on Cost Escalations in Defense Procurements: Department of Defense Authorization for Appropriation for Fiscal Year 1975, Book I of 4 Books, Committee on Armed Services, House of Representatives, 93rd Congress, 1974, pp. 132-150.

[10] Refer to [2], pp. 70-71.

[11] Refer to [7], pp. 65-93, 410-133.

[12] Life Cycle System Management Model For Army Systems, Department of the Army, Washington, D.C., Pamphlet No. 11-25, May 1975.

[13] Refer to [2], p. 72.

[14] Army Battlefield Automation Interoperability System Engineering Management Plan, Fort Monmouth, New Jersey, November 1978.

[15] Salisbury, Alan, B., "Controlling the Software Life Cycle," Darcom Computer Software Conference, Gibbs Hall, Fort Monmouth, New Jersey, November 6-7-8, 1978.

[16] Army Battlefield Interface Concept 1978,

[17] Proceedings Second U. S. Army Software Symposium, Sponsored by the U.S. Army Computer Systems Command, Williamsburg, Virginia, October 1978.

[18] Refer to [2], p. 97.

[19] Refer to [2], pp. 1248-1254, 1428-1429, 1398-1401, 1780-1812.

[20] Sapolsky, H. M. The Polaris System Development: Bureaucratic and Programmatic Success in Government, Harvard University Press, Cambridge, Ma., 1972.

[21] Hearings on Military Posture and H. R. 11500: Department of Defense Authorization for Appropriations for Fiscal Year 1977, Part 5 of 5 Parts. Committee on Armed Services, House of Representatives, 94th Congress, 2nd Session, February-March, 1976, pp. 917-987.

[22] Hearings on Military Posture and H. R. 3698: Department of Defense Authorization for Appropriations for Fiscal Year 1976 and 1977, Part 4 of 4 Parts. Committee on Armed Services, House of Representatives, 94th Congress, Ist Session, March-April 1975, pp. 3982-3984.

[23] Extract from Report of the Commission on Government Procurement, 1972. Acquisition of Major Systems, Printed March 1975.

[24] Kossiakoff, A., et al. "DoD Weapon Systems Software Management Study," Johns Hopkins University Applied Physics Laboratory, APL/JHU SR75-3, June 1975.

[25] Asch, A., et al. "DoD Weapon Systems Software Acquisition and Management Study Volume 1, MITRE Findings and Recommendations," Volume 1, MTR-6908, May 1975.

[26] W. Hillsman, "Twenty Commandments Of Software Acquisition," Presentation Given At The Software Management Conference, Washington, D.C., 1975

[27] Lieblein, E. "Problems in Software Development," Keynote Address Presented to the Joint Services Electronics Program Topical Review in Information Services, University of Illinois, 16 October 1973.

[28] "CENTACS: Information Brochure," U.S. Army for Tactical Computer Sciences, October 1975.

[29] Gansler, J. S. "Remarks," before the Polytechnic Institute of NY, Microwave Research Institute, Symposium on Computer Software Engineering, NY, NY, News Release from Office of Assistant Secretary of Defense, Washington, D.C., April 20, 1976.

[30] R & D Technology Panel, "Proposed Computer Resource Technology Objectives," as modified by DDR&E, July 12, 1976.

[31] Post-Deployment Software Support (PDSS) Management Plan, Draft, Fort Monmouth, New Jersey, 19 March 1979.

[32] Hearings on Military Posture and H. R. 5068: Department of Defense Authorization for Appropriations for Fiscal Year 1978, Part 3 of 6 Parts, Book 2 of 2 Books. Committee on Armed Services, House of Representatives, 95th Congress, pp. 1828-1829.

[33] Hearings Before Committee on Armed Services: Authorization for Military Procurement, Research, and Development, U.S. Senate, 95th Congress, Fiscal Year 1978, Part 8. ("Defense Science Board, Report of Task Force on Testing and Evaluation Policies"), February 17, 1977, pp. 1243-1256.

[34] M. Hamilton & S. Zeldin, "The Relationship Between Design and Verification," The Journal Of Systems And Software, Elsevier North Holland, Inc. New York, New York, Volume 1, No. 1, 1979,

[35] C. G. Davis and C. R. Vick, "The Software Development System," Proceedings 2nd International Conference on Software Engineering, October 1976, Addendum pp. 27-42.

[36] D. Ross, "Structured Analysis (SA): A Language for Communicating Ideas, IEEE Transactions on Software Engineering, SE-3(1) 16-34, 1977.

[37] C. A. R. Hoare, "An Axiomatic Approach to Computer Programming," CACM, 12, 576-580, 1969.

[38] Liskov, B. H. & Valdis Berzins, "An Appraisal of Program Specifications," Research Directions In Software Technology, Peter Wegner, editor, The MIT Press, Cambridge, Ma. 1979, pp. 276-301.

[39] Liskov, B. H. and Zilles, S. W., "Specification Techniques for Data Abstractions," IEEE Transactions on Software Engineering, Volume I No. 1, pp. 7-19, March, 1975.

[40] J. V. Guttag, E. Horowitz and D. Musser, "Some Extensions to Algebraic Specifications, in Proceedings of an ACM Conference on Language Design for Reliable Software, D.B. Wortman, editor, Raleigh, North Carolina, Association for Computing Machinery, New York, March, 1977.

[41] The Application Of HOS to PLRS, Higher Order Software, Inc. Technical Report #12, Higher Order Software, Inc., Cambridge, Ma., November 1977.

[42] M. Hamilton & S. Zeldin, "Higher Order Software - A Methodology For Defining Software," IEEE Transactions on Software Engineering SE-2 (1), 9-32, 1976.

[43] M. Hamilton & S. Zeldin, "The Manager As An Abstract Systems Engineer," Digest Of Papers, Fall COMPCON 77, Washington, D.C., IEEE Computer Society Cat. No. 77CH1258-3C, September 1977.

[44] M. Hamilton & S. Zeldin, AXES Syntax Description, Higher Order Software, Inc. Technical Report #4, Higher Order Software, Inc., Cambridge, Ma., December 1976.

[45] R. Hackler, From Control Maps To IDEF, Higher Order Software Inc. Technical Report In Preparation, Higher Order Software, Inc, Cambridge, Ma., 1979.

[46] J. Rood, T. To and D. Harel, "A Universal Flowcharter," Proceedings of the NASA/AIAA Workshop on Tools for Embedded Computer Systems Software, Hampton, Virginia, November 708, 1978, pp. 41-44.

[47] B. Kuhns, On Decision Support, Higher Order Software, Inc. Technical Report In Preparation, Higher Order Software, Inc., Cambridge, Ma., 1979.

[48] B. Kuhns, Modification Of Relocation Approach, Higher Order Software, Inc. DCPA Memo No. 14, Higher Order Software, Inc., Cambridge, Ma., 1979.

[49] S. Cushing, Geographically Distributed Systems In Higher Order Software, Higher Order Software, Inc. DCPA Memo No. 7, In Preparation, Higher Order Software, Inc., Cambridge, Ma., 1979.

[50] D. Harel & R. Pankiewicz, The Universal Flowcharter, Higher Order Software, Inc. Technical Report #11, Higher Order Software, Inc., Cambridge, Ma., November 1977.

[51] R. Hackler & A. Samarov, Specification For A Radar Scheduling Algorithm, Higher Order Software, Inc., Technical Report In Preparation, Higher Order Software, Inc., Cambridge, Ma, 1979.

[52] S. Cushing, Toward A Unified Semantics For The Syntaxes Of ICAM, Higher Order Software, Inc., Cambridge, Ma., May 1979.

[53] S. Cushing, "Four Models For The Description Of Systems," Axiomatic Analysis, Higher Order Software, Inc. Technical Report 19, Higher Order Software, Inc., Cambridge, Ma., September 1978.

[54] M. Hamilton & S. Zeldin, Integrated Software Development System/ Higher Order Software Conceptual Description (ISDS/HOS), Higher Order Software, Inc. Technical Report #3, Higher Order Software, Inc., Cambridge, Ma., June 1977.

[55] M. Hamilton & S. Zeldin, "ICAM System User Interface Requirements," Presentation given at the ICAM Task V First Meeting, Sheraton Hotel, Boston, Ma., December 1978.

[56] OMB Circular A-109, Major System Acquisitions, Office Of Federal Procurement Policy, OMB, New Executive Office Building, Washington, D.C., 1976.

[57] Miller, James, G., Living Systems, McGraw-Hill Book Company, New York, New York, 1978.

[58] Major System Acquisitions: A Discussion of the Application of OMB Circular No. A-109, Executive Office Of The President Office of Management and Budget Office of Federal Procurement Policy, OFPP Pamphlet No. 1, August 1976.